PERANCANGAN BASIS DATA

Setiyowati, S.Kom., M.Kom.

Sri Siswanti, S.Kom., M.Kom.

Penerbit:
Lembaga Penelitian Dan Pengabdian Kepada Masyarakat
Universitas Dian Nuswantoro Semarang

PERANCANGAN BASIS DATA
&
PENGENALAN SQL SERVER MANAGEMENT STUDIO

Setiyowati, S.Kom., M.Kom.
Sri Siswanti, S.Kom., M.Kom.

Penerbit:
Lembaga Penelitian Dan Pengabdian Kepada Masyarakat
Universitas Dian Nuswantoro Semarang
Tahun 2021

PERANCANGAN BASIS DATA & PENGENALAN SQL SERVER MANAGEMENT STUDIO

Penulis:

Setiyowati, S.Kom., M.Kom.
Sri Siswanti, S.Kom., M.Kom.

ISBN: 978-623-96867-4-1

Penerbit :
Lembaga Penelitian dan Pengabdian Kepada Masyarakat
Universitas Dian Nuswantoro Semarang

Redaksi :
LPPM Udinus
Jl. Nakula I No.5-11
Semarang 50131
Telp: (024) 352-7261, 352-0165
Fax : (024 356-9684
E-mail : sekretariat@lppm.dinus.ac.id

Desain Sampul :
Setiyowati, S.Kom., M.Kom.
Sri Siswanti, S.Kom., M.Kom.

Pencetak :
Percetakan Universitas Dian Nuswantoro Semarang

KATA PENGANTAR

Bismillahirrohmanirrohim

Puji syukur kehadirat Allah SWT atas limpahan Rahmat dan hidayah-Nya, sehingga Buku yang berjudul Perancangan Basis Data dengan Menggunakan SQL Server Management Studio dapat penulis selesaikan. Buku ini teruntuk para pemula yang baru belajar perancangan basis data, dengan menggunakan SQL Server Management Studio.

Buku ini dapat menjadi acuan atau menjadi bahan Latihan bagi para pemula yang baru belajar menggunakan SQL Server Management Studi, mulai dari cara instalasi sampai dengan cara menggunakannya.

Tidak lupa kami ucapkan terima kasih yang sebesar-besarnya kepada:

1. STMIK Sinar Nusantara Surakarta
2. Universitas Dian Nuswantoro Semarang
3. Rekan-rekan Dosen STMIK Sinar Nusantara Surakarta
4. Dan semua rekan-rekan yang telah membantu sehingga buku ini dapat terselesaikan dengan baik.

Penulis menyadari bahwa buku ini masih banyak kekurangan, untuk itu penulis berharap kritik, saran yang membangun dari para pembaca untuk perbaikan penulisan buku selanjutnya.

Surakarta, 2021

Penulis

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BAB 1. MENGENAL DATABASE

1.1 Database

Database adalah susunan *record* data operasional lengkap dari suatu organisasi atau perusahaan, yang diorganisir dan disimpan secara terintegrasi dengan menggunakan metode tertentu dalam komputer sehingga mampu memenuhi informasi yang optimal yang dibutuhkan oleh para pengguna. Database (basis data) atau dengan sebutan pangkalan data ialah suatu kumpulan sebuah informasi yang disimpan didalam sebuah perangkat komputer secara sistematik sehingga dapat diperiksa dengan menggunakan suatu program komputer agar dapat informasi dari basis data tersebut. Perangkat lunak yang digunakan untuk mengelola dan memanggil query basis data disebut dengan sistem manajemen basis data (*Database Management System*, DBMS).

Basis data istilah ini berawal dari ilmu komputer, walaupun kemudian artinya semakin luas memasukkan hal-hal diluar bidang elektronika. Untuk kesamaan pada basis data ini sebenarnya sudah ada sebelum revolusi industri yakni dalam bentuk buku besar, kuitansi dan kumpulan data yang berhubungan dengan bisnis. (Yuliansyah et al., 2014)

1.2 Komponen Sistem Basis Data (Database)

Basis data merupakan sistem yang terdiri atas kumpulan file atau tabel yang saling berhubungan dan *Database Management System* (DBMS) yang memungkinkan beberapa pemakai untuk mengakses dan manipulasi file-file tersebut.
Dalam Sistem Basis data memiliki beberapa komponen yaitu:

- Perangkat Keras (*Hardware*) Perangkat keras yang biasanya terdapat dalam sistem basis data adalah memori sekunder har*disk*.
- Sistem Operasi (*Operating System*) Sistem Operasi (*Operating System*) merupakan program yang mengaktifkan atau mengfungsikan sistem

komputer, meng*end*alikan seluruh sumber daya (*resource*) dan melakukan operasi-operasi dalam komputer. Sistem Operasi yang banyak digunakan seperti: MS-DOS, MS-Windows 95 MS Windows NT, dan Unix.

- Basisdata (Database) dapat memiliki beberapa basis data. Setiap basis data dapat berisi atau memiliki sejumlah objek basis data seperi file atau tabel. *Database Management System* (DBMS) Pengolahan basis data secara fisik tidak dilakukan oleh pemakai secara langsung, tetapi ditangani oleh sebuah perangkat lunak yang disebut DBMS yang menentukan bagaimana data disimpan, diubah dan diambil kembali.
- Pemakai (*User*) Bagi pemakai dapat berinteraksi dengan basis data dan memanipulasi data dalam program yang ditulis dalam bahasa pemograman.

Konsep dasar dari database adalah kumpulan dari catatan-catatan, atau potongan dari pengetahuan. Sebuah database memiliki penjelasan terstruktur dari jenis fakta yang tersimpan di dalamnya: penjelasan ini disebut skema. Skema menggambarkan obyek yang diwakili suatu database, dan hubungan di antara obyek tersebut. Ada banyak cara untuk mengorganisasi skema, atau memodelkan struktur database ini dikenal sebagai database model atau model data.

Model yang umum digunakan sekarang adalah model relasional, yang menurut istilah yaitu mewakili semua informasi dalam bentuk tabel-tabel yang saling berhubungan dimana setiap tabel terdiri dari baris dan kolom (definisi yang sebenarnya menggunakan terminologi matematika). Dalam model ini, hubungan antar tabel diwakili dengan menggunakan nilai yang sama antar tabel. Model yang lain seperti model hierarkis dan model jaringan menggunakan cara yang lebih eksplisit untuk mewakili hubungan antar tabel.

Konsep dasar dari basis data ialah kumpulan dari sebuah catatan atau sebuah potongan dari pengetahuan. Sebuah basis data memiliki penjelasan testruktur dari jenis fakta yang tersimpan di dalamnya, penjelasan tersebut dengan skema. Skema menggambarkan sebuah objek yang diwakili suatu basis data dan memiliki hubungan diantara objek tersebut. Ada banyak cara untuk mengorganisasi skema atau memodelkan struktur basis data, ini dikenal sebagai model basis data atau model data. Biasanya model yang umum digunakan sekarang ialah model

relasional yang mewakili semua informasi dalam bentuk tabel-tabel yang saling berhubungan dimana setiap tabel terdiri dari baris dan kolom (definisi yang sebenarnya menggunakan terminologi matematika). Dalam model ini hubungan antar tabel diwakili dengan menggunakan nilai yang sama antar tabel. Model yang lain seperti model hierarkis dan model jaringan menggunakan cara yang lebih eksplisit untuk mewakili hubungan antar tabel.

1.3 Perangkat Membuat Database

Database dapat dibuat dan diolah dengan menggunakan suatu program komputer, yaitu yang biasa kita sebut dengan *software* (perangkat lunak). *Software* yang digunakan untuk mengelola dan memanggil kueri (*query*) database disebut *Database Management System* (DBMS) atau jika diterjemahkan kedalam bahasa Indonesia berarti "Sistem Manajemen Basis Data". (Raharjo, 2012)

DBMS terdiri dari dua komponen, yaitu *Relational Database Management System* (RDBMS) dan *Overview of Database Management System* (ODBMS). RDBMS meliputi *Interface Drivers, SQL Engine, Transaction Engine, Relational Engine*, dan *Storage Engine*. Sedangkan ODBMS meliputi *Language Drivers, Query Engine, Transaction Engine*, dan *Storage Engine*.(Kalaivani & Shyamala, 2016)

Sedangkan untuk lavel dari softwarenya s*end*iri, terdapat dua *level software* yang memungkinkan kita untuk membuat sebuah database antara lain adalah *High Level Software* dan *Low Level Software*. Yang termasuk di dalam *High Level Software*, antara lain seperti *Microsoft SQL Server, Oracle, Sybase, Interbase, XBase, Firebird, MySQL, PostgreSQL, Microsoft Access*, dBase III, *Paradox, FoxPro, Visual FoxPro, Arago, Force, Recital, dbFast, dbXL, Quicksilver, Clipper, FlagShip, Harbour, Visual dBase*, dan *Lotus Smart Suite Approach*. Sedangkan yang termasuk di dalam *Low Level Software* antara lain *Btrieve* dan *Tsunami Record Manager.*

1.4 Jenis Tipe Database

Terdapat 12 tipe database, antara lain *Operational database, Analytical database, Data warehouse, Distributed database, End-user database, External database, Hypermedia databases on the web, Navigational database, In-memory databases, Document-oriented databases, Real-time databases, dan Relational Database*. 12 tipe database tersebut dijelaskan sebagai berikut:

1. *Operational Database Database* ini menyimpan data rinci yang diperlukan untuk m*end*ukung operasi dari seluruh organisasi. Mereka juga disebut *subject-area databases* (SADB), transaksi database, dan produksi database. **Contoh:** database pelanggan, database pribadi, database inventaris, akuntansi database.
2. *Analytical database*, database ini menyimpan data dan informasi yang diambil dari operasional yang dipilih dan eksternal database. Mereka terdiri dari data dan informasi yang dirangkum paling dibutuhkan oleh sebuah organisasi manajemen dan *End-user* lainnya. Beberapa orang menyebut analitis multidimensi database sebagai database, manajemen database, atau informasi database.
3. *Data warehouse*, sebuah data *warehouse* menyimpan data dari saat ini dan tahun-tahun sebelumnya – data yang diambil dari berbagai database operasional dari sebuah organisasi. Data *warehouse* menjadi sumber utama data yang telah diperiksa, diedit, standar dan terintegrasi sehingga dapat digunakan oleh para manajer dan pengguna akhir lainnya di seluruh organisasi profesional. Perkembangan terakhir dari data *warehouse* adalah dipergunakan sebagai *Shared nothing architecture* untuk memfasilitasi *extreme scaling*.
4. *Distributed database* adalah database kelompok kerja lokal dan departemen di kantor regional, kantor cabang, pabrik-pabrik dan lokasi kerja lainnya. Database ini dapat mencakup kedua segmen yaitu operasional dan *user* database, serta data yang dihasilkan dan digunakan hanya pada pengguna situs s*end*iri.

5. *End-user database,* database ini terdiri dari berbagai file data yang dikembangkan oleh *end-user* di *workstation* mereka. **Contoh**: koleksi dokumen dalam *spreadsheet*, *word processing* dan bahkan *download* file.
6. *External database,* database ini menyediakan akses ke eksternal, data milik pribadi online tersedia untuk biaya kepada pengguna akhir dan organisasi dari layanan komersial. Akses ke kekayaan informasi dari database eksternal yang tersedia untuk biaya dari layanan online komersial dan dengan atau tanpa biaya dari banyak sumber di Internet.
7. *Hypermedia databases on the web* adalah kumpulan dari halaman-halaman multimedia yang saling berhubungan di sebuah situs web. Mereka terdiri dari home page dan halaman hyperlink lain dari multimedia atau campuran media seperti teks, grafik, gambar foto, klip video, audio dll.
8. *Navigational database,* dalam navigasi database, *queries* menemukan benda terutama dengan mengikuti referensi dari objek lain.
9. *In-memory databases,* database di memori terutama bergantung pada memori utama untuk penyimpanan data komputer. Ini berbeda dengan sistem manajemen database yang menggunakan *disk* berbasis mekanisme penyimpanan. Database memori utama lebih cepat daripada dioptimalkan *disk* database sejak Optimasi algoritma internal menjadi lebih sederhana dan lebih sedikit CPU mengeksekusi instruksi. Mengakses data dalam menyediakan memori lebih cepat dan lebih dapat diprediksi kinerja dari *disk*. Dalam aplikasi di mana waktu respon sangat penting, seperti peralatan jaringan telekomunikasi yang mengoperasikan sistem darurat, database memori utama yang sering digunakan.
10. *Document-oriented databases* merupakan program komputer yang dirancang untuk aplikasi berorientasi dokumen. Sistem ini bisa diimplementasikan sebagai lapisan di atas sebuah database relasional atau objek database. Sebagai lawan dari database relasional, dokumen berbasis database tidak menyimpan data dalam tabel dengan ukuran seragam kolom untuk setiap *record*. Sebaliknya, mereka menyimpan setiap catatan sebagai dokumen yang memiliki karakteristik tertentu. Sejumlah bidang panjang apapun dapat

ditambahkan ke dokumen. Bidang yang dapat juga berisi beberapa bagian data.

11. *Real-time databases* adalah sistem pengolahan dirancang untuk menangani beban kerja negara yang dapat berubah terus-menerus. Ini berbeda dari database tradisional yang mengandung data yang terus-menerus, sebagian besar tidak terpengaruh oleh waktu. Sebagai contoh, pasar saham berubah dengan cepat dan dinamis. *Real-time processing* berarti bahwa transaksi diproses cukup cepat bagi hasil untuk kembali dan bertindak segera. *Real-time database* yang berguna untuk akuntansi, perbankan, hukum, catatan medis, multi-media, kontrol proses, sistem reservasi, dan analisis data ilmiah.
12. *Relational Database* standar komputasi bisnis sejak tahun 2009, *relational database* adalah database yang paling umum digunakan saat ini. Menggunakan meja untuk informasi struktur sehingga mudah untuk mencari.

1.5 Variabel Database

Database mempunyai dua varian model, yaitu *model Post-relational database dan model Object database*. Masing-masing variable dijelaskan sebagai berikut:

A. *Post-Relational Database Models*

Sebuah produk yang menawarkan model data yang lebih umum dari model relasional dan dikenal sebagai post-relational. Model data dalam produk tersebut mencakup hubungan namun tidak dibatasi oleh Prinsip Informasi yang mana mewakili semua informasi dengan nilai-nilai data dalam kaitannya dengan hal itu. Sebagian dari perluasan ini ke model relasional benar-benar mengintegrasikan konsep-konsep dari teknologi yang tanggal pre-date the relational model.

Sebagai contoh, mereka mengijinkan representasi dari *directed graph* dengan *trees* pada *node*. Beberapa produk menerapkan model tersebut melakukannya dengan memperluas sistem database relasional dengan fitur

non-relasional. Sedangkan yang lainnya, telah tiba di tempat yang sama dengan menambahkan fitur relasional untuk sistem pre-relational. Anehnya, hal ini memungkinkan produk-produk yang secara historis *pre-relational*, seperti PICK dan gondok, untuk membuat klaim yang masuk akal untuk *post-relational* dalam arsitektur saat ini.

B. Object Database Models

Dalam beberapa tahun terakhir, paradigma yang berorientasi pada obyek telah diterapkan dalam bidang-bidang seperti teknik dan spasial database, telekomunikasi dan ilmu pilmiah lainnya. Para konglomerasi pemrograman berorientasi objek dan teknologi database mengarah pada model pemrograman baru yang dikenal sebagai Object database. Database ini berusaha untuk membawa dunia database dan aplikasi-dunia pemrograman lebih dekat bersama-sama, khususnya dengan memastikan bahwa database menggunakan jenis system yang sama seperti program aplikasi.

Hal ini bertujuan untuk menghindari *overhead* (kadang-kadang disebut sebagai ketidakcocokan impedansi) untuk mengkonversi informasi antara perwakilan di database (misalnya sebagai baris dalam tabel) dan perwakilan di program aplikasi (biasanya sebagai objek). Pada saat yang sama, *object database* berupaya untuk memperkenalkan ide-ide kunci dari pemrograman objek, seperti *encapsulation* dan *polymorphism*, ke dalam dunia database.

Berbagai cara-cara ini telah dicoba untuk menyimpan objek dalam database. Beberapa produk mengalami masalah dari sisi pemrograman aplikasi, dengan membuat objek dimanipulasi oleh program terus-menerus. Hal ini juga biasanya memerlukan penambahan pertanyaan semacam bahasa, karena bahasa pemrograman konvensional tidak menyediakan fungsionalitas tingkat bahasa untuk menemukan obyek berdasarkan isi informasi mereka.

1.6 Tahapan Perancangan Database

Perancangan database (basis data) merupakan upaya untuk membangun sebuah basis data dalam suatu lingkungan bisnis, untuk membangun sebuah basis data terdapat tahapan-tahapan yang perlu dilalui yaitu:

- Perencanaan database (basis data)
- Mendefiniskan sistem
- Analisa dan mengumpulkan kebutuhan
- Perancangan database (basis data)
- Perancangan aplikasi
- Membuat *prototype*
- Implementasi
- Konversi data
- Pengujian
- Pemeliharaan operasional

A. Perangkat Lunak Database (Basis Data)

Perangkat lunak database (basis data) yang banyak digunakan dalam pemrograman yaitu:

a. MySQL

MySQL adalah sebuah perangkat lunak pada sistem manajemen basis data SQL atau DBMS (*database management system*) yang *multithread*, multi *user*, dengan sekitar 6 juta instalasi diseluruh dunia. MySQLAB membuat MySQL tersedia sebagai perangkat lunak gratis dibawah lisensi GNU *General Public License* (GPL) tetapi mereka juga menjual dibawah lisensi komersial untuk kasus-kasus dimana penggunaanya tidak cocok dengan penggunaan GPL. Tidak sama dengan proyek-proyek seperti Apache dimana perangkat lunak dikembangkan oleh komunitas umum, dan hak cipta untuk kode sumber dimiliki oleh penulisnya masing-masing, MySQL dimiliki dan disponsori oleh

sebuah perusahaan komersial Swedia MySQL AB, dimana memegang hak cipta hamper atas semua kode sumbernya. Kedua orang swedia dan satu finlandia yang m*end*irikan MySQL AB ialah David Axmark. Allan Larson dan Michael Monty Widenius.

b. Microsoft SQL Server

Sebuah sistem manajemen basis data relaisonal (RDBMS) produk Microsoft. Bahasa kueri utamanya ialah Transact-SQL yang merupakan implementasi dari SQL standar ANSI/ISO yang digunakan oleh Microsoft dan Sybase. Yang pada Umumnya SQL Server digunakan didunia bisnis yang memiliki basis data berskala kecil hingga menengah, tetapi kemudian berkembang dengan digunakannya SQL Server pada basis data besar. Microsoft SQL Server dan Sybase/ASE dapat berkomunikasi lewat jaringan dengan menggunakan protocol TDS (*Tabular Data Stream*). Selain dari itu Microsoft SQL Server juga m*end*ukung ODBC (*Open Database Connectivity*) dan mempunyai driver JDBC untuk bahasa pemrograman Java. Fitur yang lain dari SQL Server ini adalah kemampuannya untuk membuat basis data *mirroring* dan *clustering*. Pada versi sebelumnya MS SQL Server 2000 terserang oleh *cacing computer SQL Slammer* hal tersebut mengakibatkan kelambatan pada akses internetnya.

c. *Relational Database Management System* (RDBMS)

MySQL adalah *Relational database management system* (RDBMS) yang didistribusikan secara gratis dibawah lisensi GPL (*General Public License*) dimana setiap orang bebas untuk menggunakan MySQL namun tidak boleh dijadikan produk turunan yang bersifat komersial. MySQL sebenarnya merupakan turunan salah satu konsep utama dalam database sejak lama, terutama untuk pemilihan atau seleksi dan pemasukan data yang memungkinkan pengoperasian data dikerjakan dengan mudah secara otomatis. Keandalan suatu database (DBMS) dapat diketahui dari cara kerja optimizernya dalam melakukan proses perintah-perintah SQL yang dibuat oleh

user maupun program-program aplikasinya. Sebagai database server, MySQL dapat dikatakan lebih unggul dibandingkan database server lainnya dalam query data. Hal ini terbukti untuk *querty* yang dilakukan oleh *single user*, kecepatan *query MySQL* bias sepuluh kali lebih cepat dari *postgreSQL* dan lima kali lebih cepat dibandingkan interbase.

d. Clipper

Merupakan bahasa pemrograman computer keluarga XBase yang digunakan untuk membuat program komputer utamanya yang berjalan pada sistem operasi DOS. Secara lebih spesifik, clipper umumnya digunakan untuk membuat program-program yang terkait dengan database/bisnis misalnya manajemen simpan/pinjam, akuntansi dan lain-lain. Sejarah clipper pertama kali diperkenalkan pada tahun 1985 oleh Nantucket yang kemudian dijual kepada *Computer Assosiates* sebagai compiler untuk Dbase III yang sangat popular pada masa itu. Kompilasi kode-kode Dbase berarti mengubahnya dari kode interpretasi (kode sumber yang bias dibaca oleh manusia) yang harus interpretasikan oleh komputer setiap kali setiap baris dijalankan, menjadi P-code (*pseudo-code*) yang menggunakan mesin virtual untuk memproses p-code yang telah dikompilasi tersebut. Meskipun p-code tidak lebih cepat dari pada kode mesin yang dihasilkan oleh kompiler bahasa bahasa lain (C++), namun secara kesuluruhan p-code masih jauh lebih cepat dibandingkan interpreter.

e. DBASE

Sebuah *System Manajemen Basisdata* (DBMS) yang secara luas digunakan pada mikrokomputer yang dikenalkan oleh Ashton-Tate untuk computer CP/M dan kemudian untuk platform Apple II, Apple Macintosh dan IBM PC dengan DOS yang menjadi salah satu perangkat lunak yang paling laris selama beberapa tahun pada saat itu. Ketidakmampuan dBASE untuk bertrasisi dengan operasi yang lebih baru, Microsoft Windows pada akhirnya membuat penggunaan dBASE tergantikan pleh produk-produk lainnya yang lebih baru seperti Paradox, Clipper, Foxpro dan Microsoft Access.

Kepemilikan Dbase pada akhirnya dijual ke Borland pada tahun 1991 dan pada tahun 1999 Borland menjaual hak atas jajaran produk dBASE pada sebuah perusahaan baru Dbase Inc.Dimulai dari pertengahan tahun 1980-an banyak v*end*or membuat dialek ataupun variasi pada produk mereka ataupun pada bahasanya s*end*iri. Termasuk didalamnya FoxPro (sekarang dikenal sebagai Visual FoxPro), Quicksilver, Clipper, Xbase ++, Flagship, dan Harbour. Mereka-mereka inilah yang secara informasi dikenal atau disebut sebagai xBase atau Xbase. Dasar file format dBase yang dikenal sebagai file.dbf, saat ini merupakan salah satu format yang luas digunakanoleh banyak aplikasi yang membutuhkan format sederhana untuk menyimpan data-data secara terstruktur. Dbase dilinsensikan pada penggunanya untuk jangka waktu lima tahun dalam masa yang tidak mungkin bagi pengguna untuk mengoperasikan Dbase selama jangka waktu tersebut.

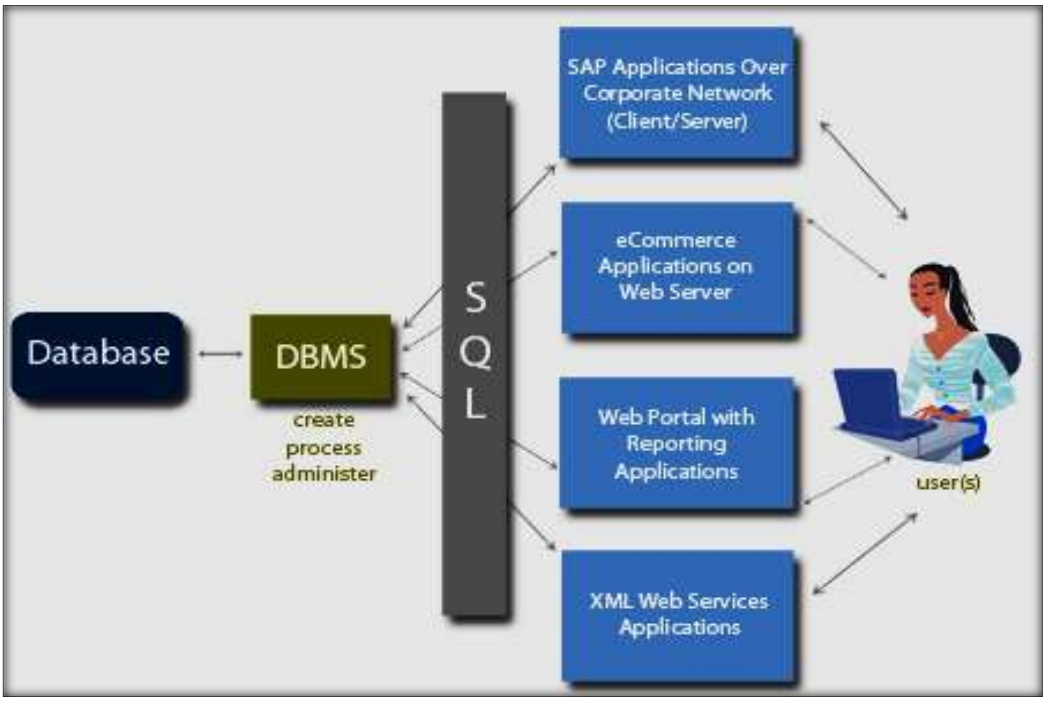
f. *Firebird*

Firebird atau disebut juga *FirebirdSQL* ialah sistem manajemen basidata relasional yang menawarkan fitur-fitur yang terdapat dalam standar ANSI SQL-99 dan SQL-2003. RDBMS ini berjalan baik di Linux, Windows maupun pada sejumlah *platform Unix*. *Firebird* di arahkan dan di-maintain oleh *FirebirdSQL Foundation*. Ia merupakan turunan dari Interbase versi open source milik Borland. Modul-modul kode baru ditambahkan pada *Firebird* dan berlisensi dibawah *Initial Developer's Public License* (IDPL) sementara modul-modul aslinya dirilis oleh Inprise berlisensi dibawah *InterBase Public License1.0*. kedua lisensi tersebut merupakan versi modifikasi dari *Mozilla Public License 1.1*.

B. *Data Base Management System* (DBMS)

Data Base Management System adalah kumpulan program yang digunakan untuk m*end*efinisikan, mengatur dan memproses database, sedangkan database adalah sebuah struktur yang dibangun untuk menyimpan data. DBMS adalah *tools*

yang berperan dalam membangun struktur. Contoh aplikasi DBMS: My SQL, Oracle IBM, DB2 dll.

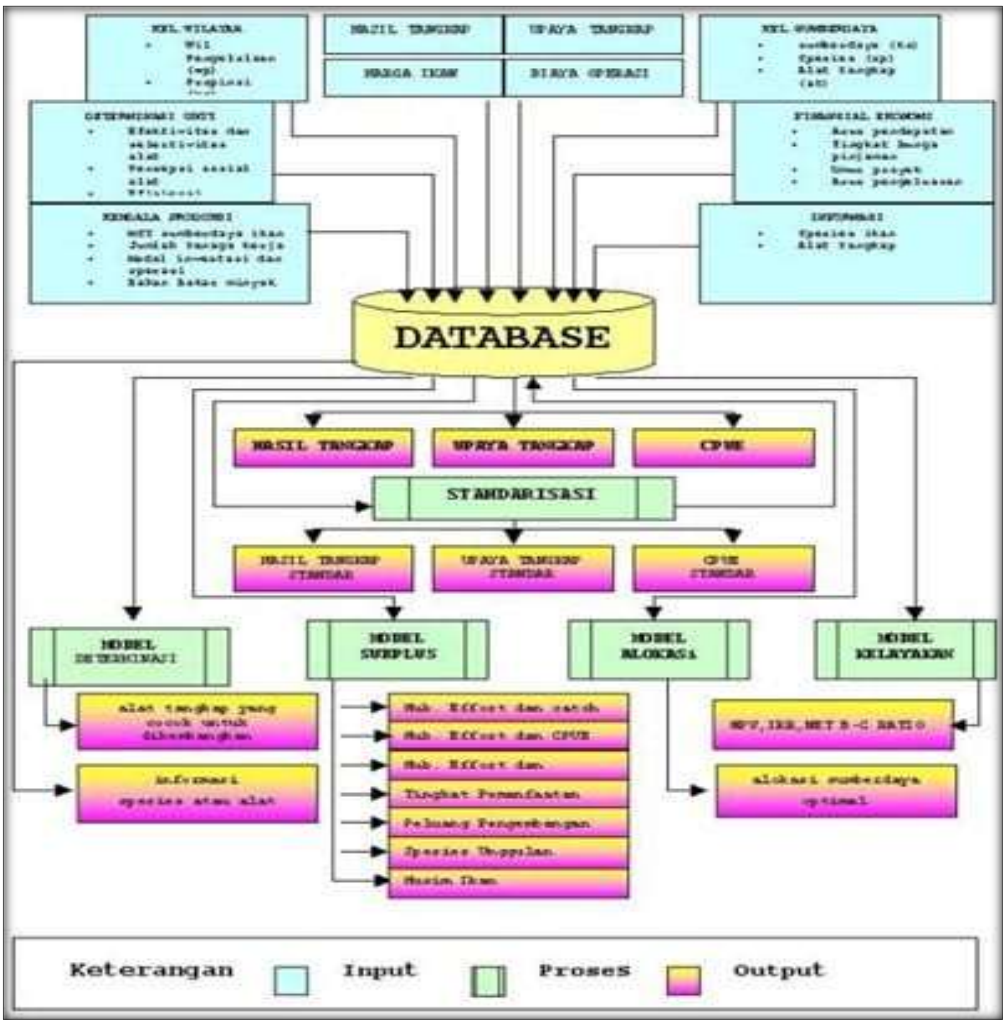


Gambar 1. *Data Base Management System*

C. Fungsi *Data Base Management System* (DBMS)

Diperlukan suatu sistem untuk diintegrasikan data file kedalam suatu file sehingga bisa melayani *user* yang berbeda. perangkat keras dan perangkat lumak serta prosedur yang mengelola data base manajemen sistem. Fungsi sistem manajemen basis data adalah:

1. Menyediakan sistem akses cepat.
2. Mengurangi kerangkapan data dan redudancy data.
3. Memungkan adanya updating secara bersana.
4. Menyediakan sistem yang memungkinkan dilakukan pengembangan database.
5. Memberikan perlindungan dari pihak pemakai tidak berhak.



Gambar 2.. Fungsi Sistem Manajemen Basis Data

D. Tabel

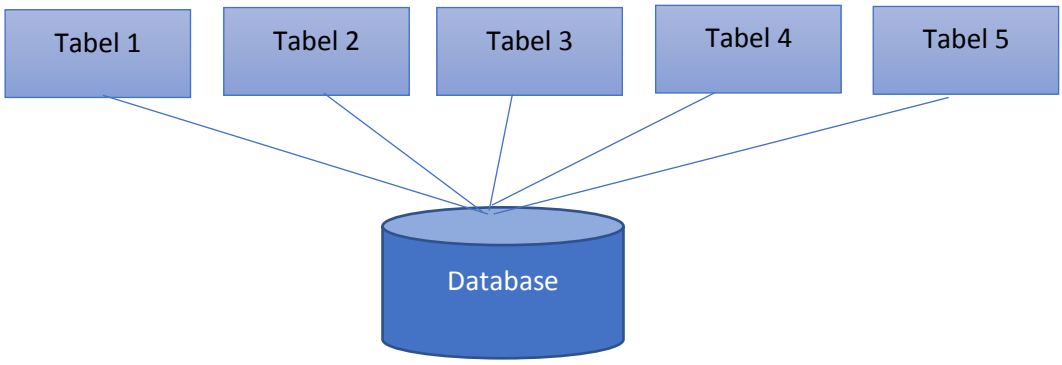
Data dalam database akan diklasifikasikan berdasarkan jenisnya dan disimpan di dalam wadah tersendiri yang disebut dengan Tabel. Database disebut juga dengan kumpulan tabel. Tabel adalah suatu entitas yang tersusun atas field dan *record*. Contoh tabel seperti pada Tabel 1.

Tabel 1. Contoh Tabel

Kode Barang	Nama Barang
B001	Buku Tulis
B002	Buku Gambar
C001	Bolpoin
C002	Pensil

Tabel 1 terdiri atas Field Kode Barang dan Nama Barang, sedangkan jumlah baris sebanyak 4 data atau 4 *record*.

Dalam sebuah model relasional, database tersusun atas beberapa tabel yang saling terhubung atau memiliki keterkaitan satu sama lainnya. Gambar 2 merupakan ilustrasi dari database:



Gambar 3. Ilustrasi Database

Ilustrasi database pada Gambar 2 menunjukkan bahwa database terdiri dari 5 tabel yang saling terhubung atau relasional, masing-masing table harus memiliki relasi. Relasi antar tabel dibentuk menggunakan *field* yang terdapat dalam tabel-tabel yang bersangkutan, melalui pendefinisian *constraint* (*primary key* dan *foreign key*).

E. *Contraint*

Constraint adalah aturan yang mendefinisikan nilai atau data yang dapan disimpan di dalam database, baik melalui operasi *INSERT*, *UPDATE* atau *DELETE*.

F. Perbedaan *Primary Key, Foreign Key dan Candidate Key*

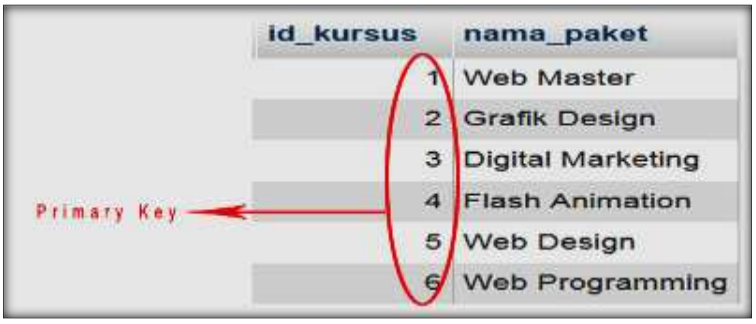
Key pada SQL merupakan gabungan beberapa atribut dimana fungsinya adalah untuk membedakan semua basis data didalam tabel secara unik ataupun suatu cara untuk menghubungkan antara tabel satu dengan tabel yang lainnya. Didalam SQL, *key* terbagi menjadi beberapa jenis diantaranya adalah sebagai berikut:

a. *Primary Key*
b. *Foreign Key*
c. *Candidate Key*
d. *Super Key*
e. *Alternate Key*
f. *Composite Key*

Dari semua *key* diatas, saya akan membahas apa saja perbedaan dari masing-masing *key* tersebut, namun kali ini saya akan membahas terlebih dahulu Perbedaan *Primary Key*, *Foreign Key* dan *Candidate Key*.

a. *Primary Key*

Primary Key merupakan sebuah aturan dimana fungsinya adalah untuk membedakan anatara baris satu dengan baris lainnya yang ada pada tabel dan bersifat unik. Gambar 4 adalah contoh *primary key* pada salah satu tabel.



Gambar 4. Contoh *Primary Key*

Ada ketentuan yang harus diperhatikan ketika *field* yang menjadi *primary key* yakni:

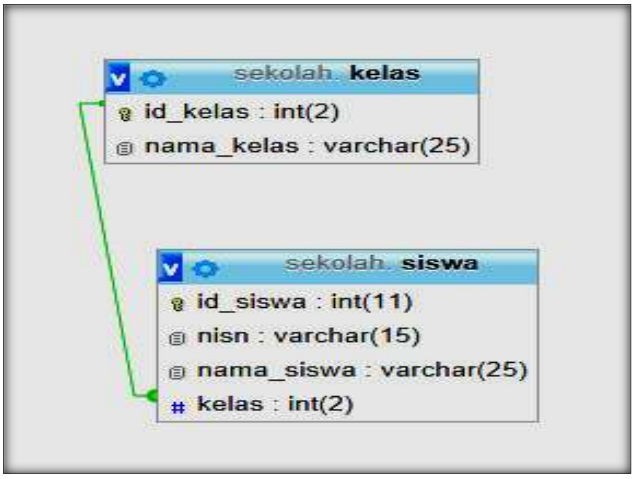
- Data tidak boleh sama atau ganda (unik).
- Data tidak boleh bernilai *null*.

Contoh sederhana penerapan *primary key* seperti contoh diatas adalah id.

b. *Foreign Key*

Dari namanya kita bisa mengira bahwa *foreign* (tamu) *key*, merupakan suatu atribut untuk melengkapi hubungan yang menunjukan ke induknya, itu artinya *field* pada tabel merupakan kunci tamu dari tabel lain. Dan biasanya penggunaan *foreign key* akan sangat dibutuhkan ketikan kita menemukan banyak tabel dan ingin menghubungkan satu tabel dengan tabel lainnya.

Contoh *Foreign key* seperti pada Gambar 5.



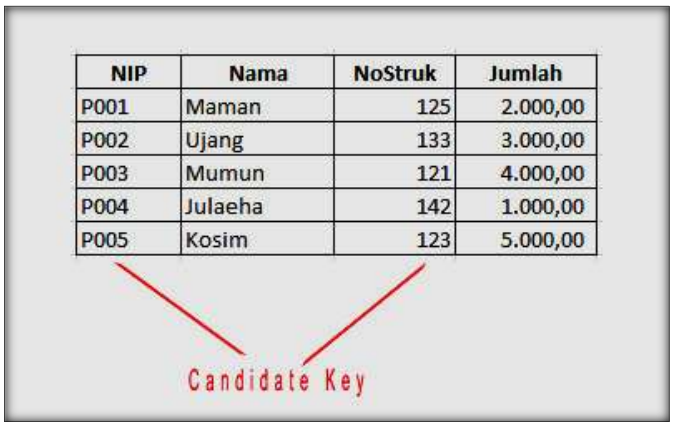
Gambar 5. *Foreign Key*

Aturan dalam p*end*efinisian *foreign key*:

1. satu tabel dapat memiliki lebih dari satu *foreign key*.
2. kolom yang diacu harus didefinisikan sebagai *primary key* atau *unique.*
3. *foreign key* tidak bersifat unik.

c. *Candidate Key*

Yang terakhir dari pembahasan *key* pada SQL adalah *candidate key*, *candidate key* merupakan suatu atribut ataupun *super key* yang mengidentifikasi secara unik untuk kejadian spesifik dari entitas. Gambar 6 adalah contoh *candidate key*.



Gambar 6. Contoh *Candidate Key*

Unique

Fungsi *unique* sama seperti *primary key*, yaitu untuk memastikan bahwa baris data terdapat dalam suatu tabel bersifat unik (tidak sama). *Unique key* diizinkan memasukkan nilai *null*.

Check

Constraint ini berfungsi untuk membatasi nilai-nilai yang dapat dimasukkan ke dalam suatu kolom dalam tabel. Contoh jenis kelamin nilainya dapat dibatasi hanya berupa PRIA atau WANITA saja, selain nilai itu server database akan menolak.

Indeks

Indeks adalah *object database* yang berfungsi untuk mempercepat proses pengambilan, pengurutan maupunpencarian data pada suatu tabel di dalam database. Data pada tabel yang sudah diindeks akan diurutkan berdasarkan kolom indeks, sehingga proses pencarian lebih mudah.

G. Model Database

Model database adalah suatu konsep yang terintegrasi dalam menggambarkan

hubungan (*relationships*) antar data dan batasan-batasan (*constraint*) data dalam suatu sistem database. Model data yang paling umum, berdasarkan pada

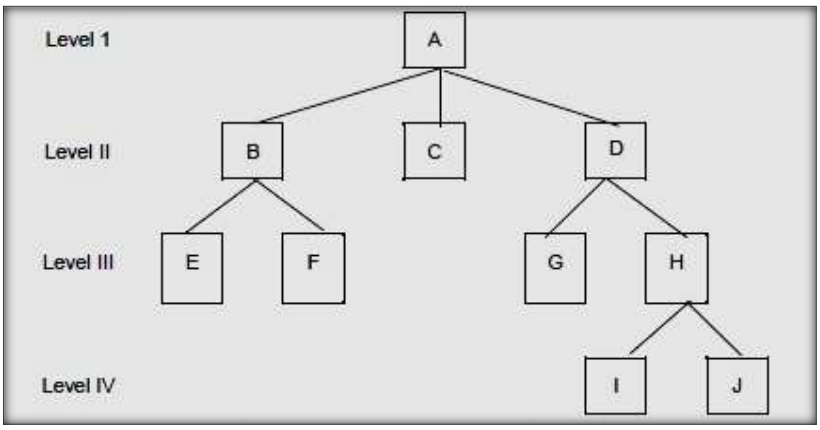
bagaimana hubungan antar *record* dalam database (*Record Based Data Models*), terdapat tiga jenis, yaitu:

a. Model Database Hirarki (*Hierarchical Database Model*)
b. Model Database Jaringan (*Network Database Model*)
c. Model Database Relasi (*Relational Database Model*)

Model database hirarki dan jaringan merupakan model database yang tidak banyak lagi dipakai saat ini, karena adanya berbagai kelemahan dan hanya cocok untuk struktur hirarki dan jaringan saja. Artinya tidak mengakomodir untuk berbagai macam jenis persoalan dalam suatu sistem database. Model database yang paling banyak dipakai saat ini adalah model database relasi, karena mampu mengakomodir berbagai permasalahan dalam sistem database. Berikut keterangan tentang model database ini.

a. Model Database Hirarki (*Hierarchical Database Model*)

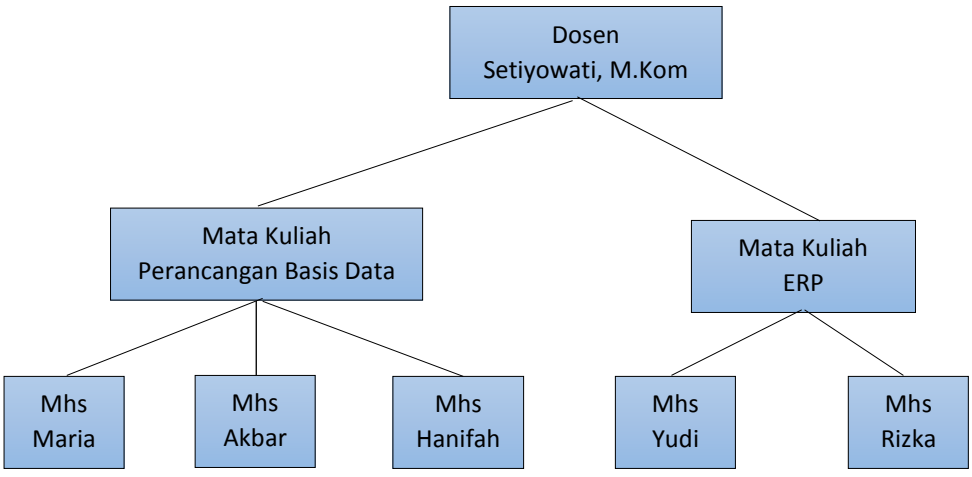
Model database hirarki disebut juga model pohon, karena hubungan antar simpul digambarkan seperti struktur pohon (*tree-structured*) yang dibalik dengan pola hubungan orang tua – anak (*parent–child*). Simpul yang paling atas disebut akar (*root*) dan paling bawah disebut daun. Setiap simpul digambarkan dengan lingkaran atau kotak. Simpul yang berada di atas simpul lainnya disebut orang tua, sedangkan yang berada di bawahnya di sebut anak, dimana seorang orang tua bisa mempunyai satu anak (jenis hubungan satu ke satu, *one to one*) atau mempunyai beberapa anak (jenis hubungan satu ke banyak, *one to many*). Tapi satu anak hanya boleh punya satu orang tua (jenis hubungan satu ke satu, one to one). Untuk jelasnya dapat dilihat pada Gambar 7.



Gambar 7. Model database hirarki

Pada Gambar 7 simpul A disebut akar dan juga bertindak sebagai orang tua dengan anak simpul A, B dan C. Simpul E, F, I dan J disebut daun, dimana E dan F merupakan anak dari simpul B serta simpul I dan J merupakan anak dari simpul H.

Dalam aplikasi nyata, dapat dilihat dalam hubungan antara dosen dengan matakuliah yang diampu serta mahasiswanya. Perhatikan Gambar 8.



Gambar 8. Contoh Database Hirarki

Kelemahan utama dari model database hirarki adalah ketidakmampuannya dalam mengelola hubungan banyak ke banyak (*many to many*), sehingga apabila ada jenis hubungan ini pada model database, maka banyaknya redundansi database tidak dapat terelakkan lagi.

Misalnya pada contoh diatas, mahasiswa merupakan anak dari simpul matakuliah, dengan pilihan ini, maka mahasiswa yang sedang cuti (istirahat kuliah) menjadi tidak tertangani, karena yang disimpan hanyalah data mahasiswa (anak) yang mengambil matakuliah (orang tua), akibatnya ada data yang hilang.

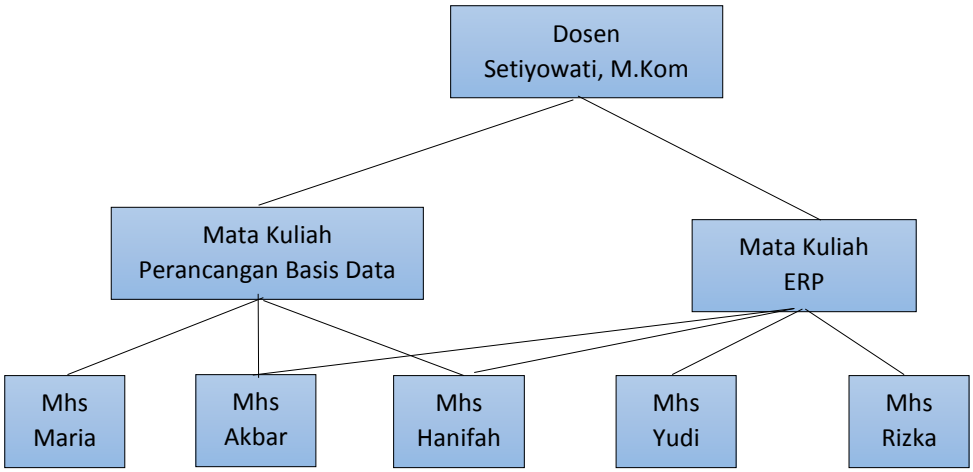
Keunggulan model database ini terletak pada keteraturan struktur yang ditunjukkannya dan hanya sangat cocok untuk sistem yang keterkaitan atau hubungan antara *record* nya mengikuti struktur hirarki. Karena keterbatasan pemakaiannya dan adanya kelemahan yang cukup mendasar, penggunaan model database ini dalam pengelolaan sistem database sudah ditinggalkan.

Simpul B disebut anak dari simpul A, tapi disisi lain simpul B juga merupakan orang tua dengan anak simpul E dan F.

b. Model Database Jaringan (*Network Database Model*)

Model database jaringan merupakan pengembangan dari model database hirarki, dimana kelemahan yang ada pada model database hirarki yaitu ketidakmampuannya dalam mengelola hubungan banyak ke banyak (*Many to Many*) telah dapat diatasi dengan model database jaringan ini. Dalam model ini, data di representasikan sebagai koleksi *record* dan hubungan antar *record* direpresentasikan sebagai *pointer*. Oleh karena itu, model database jaringan mampu menyatakan hubungan:

- Satu ke Satu (*One to One*, 1: 1), satu orang tua punya satu anak.
- Satu ke Banyak (*One to Many*, 1: M) Satu orang tua punya beberapa anak,
- Banyak ke Banyak (*Many to Many*, N: M), beberapa anak punya beberapa orang tua.



Gambar 9. Contoh Database Jaringan

Kelemahan dalam model database ini adalah lebih kompleks dan sulitnya dalam proses query, begitu juga halnya dalam manipulasi data yang harus dilaksanakan dengan menelusuri data pointer pada setiap *record*nya.

Kelebihan model database ini adalah dari segi efisiensi penyimpanan data, karena tidak adanya data yang duplikat (redundansi) dan akses yang cepat karena langsung memanfaatkan pointer ke alamat fisik data. Karena kompleksitas yang tinggi, apalagi diterapkan pada sistem database yang begitu kompleks, maka model database ini tidak tepat lagi untuk digunakan. Saat ini, model database jaringan sudah jarang sekali dipakai, kecuali untuk keperluan penelitian (*research*) saja.

c. Model Database Relasi (*Relational Database Model*)

Model database relasi merupakan model database yang paling banyak digunakan saat ini, karena paling sederhana dan mudah digunakan serta yang paling penting adalah kemampuannya dalam mengakomodasi berbagai kebutuhan pengelolaan database. Sebuah database dalam model ini disusun dalam bentuk tabel dua dimensi yang terdiri dari baris (*record*) dan kolom (*field*), pertemuan antara baris dengan kolom disebut item data (*data value*), tabel-tabel yang ada dihubungkan (*relationship*) sedemikian rupa menggunakan *field-field* kunci (*key field*) sehingga dapat meminimalkan duplikasi data.

BAB 2. NORMALISASI DATABASE

2.1. Normalisasi Database

Normalisasi merupakan sebuah teknik dalam logical desain sebuah basis data yang mengelompokkan atribut dari suatu relasi sehingga membentuk struktur relasi yang baik (tanpa redudansi). (Puspitasari et al., 2016)

Normalisasi adalah proses pembentukan struktur basis data sehingga sebagian besar *ambiguity* bisa dihilangkan.

Tujuan Normalisasi adalah:

- Untuk menghilang kerangkapan data.
- Untuk mengurangi kompleksitas.
- Untuk mempermudah pemodifikasian data.

Jika data dalam database tersebut belum di normalisasi maka akan terjadi 3 kemungkinan yang akan merugikan sistem secara keseluruhan.

1. *INSERT* Anomali: Situasi dimana tidak memungkinkan memasukkan beberapa jenis data secara langsung di database.
2. *DELETE* Anomali: Penghapusan data yang tidak sesuai dengan yang diharapkan, artinya data yang harusnya tidak terhapus mungkin ikut terhapus.
3. *UPDATE* Anomali: Situasi dimana nilai yang diubah menyebabkan inkonsistensi database, dalam artian data yang diubah tidak sesuai dengan yang diperintahkan atau yang diinginkan.

Proses Normalisasi

- Data diuraikan dalam bentuk tabel, selanjutnya dianalisis berdasarkan persyaratan tertentu ke beberapa tingkat.
- Apabila tabel yang diuji belum memenuhi persyaratan tertentu, maka tabel tersebut perlu dipecah menjadi beberapa tabel yang lebih sederhana sampai memenuhi bentuk yang optimal.

Pentingnya Normalisasi

Suatu rancangan database disebut buruk jika:

- Data yang sama tersimpan di beberapa tempat (file atau *record*).

- Ketidakmampuan untuk menghasilkan informasi tertentu.
- Terjadi kehilangan informasi.
- Terjadi adanya redudansi (pengulangan) atau duplikasi data sehingga memboroskan ruang penyimpanan dan menyulitkan saat proses updating data.
- Timbul adanya *NULL VALUE*.
- Kehilangan informasi bisa terjadi bila pada waktu merancang database (melakukan proses dekomposisi yang keliru).
- Bentuk normalisasi yang sering digunakan adalah 1st NF, 2nd NF, 3rd NF, dan BCNF.

2.2. Normalisasi Database, 1NF, 2 NF, 3NF

Normalisasi database terdiri dari banyak bentuk, dalam ilmu basis data ada setidaknya 9 bentuk normalisasi yang ada yaitu: 1NF, 2NF, 3NF, EKNF, BCNF, 4NF, 5NF, DKNF, dan 6NF. Namun dalam prakteknya dalam dunia industri bentuk normalisasi yang paling sering digunakan ada sekitar 5 bentuk.

A. Normal Form

Data yang direkam dan dimasukkan secara mentah dalam suatu tabel pada bentuk ini sangat mungkin terjadi inkonsistensi dan anomali data. Guna melakukan normalisasi database kita harus mengidentifikasi data seperti apa yang akan disimpan. Sebagai contoh data dari struk penjualan, seperti terlihat pada Tabel 2.

Tabel *2*. Contoh Struk Penjualan

Kode Faktur	Tanggal	Kode Barang	Nama Barang	Harga Barang	QTY
KD-1001	01/05/2021	BRG-001	Supermie Goreng	2500	8
		BRG-002	Supermie Rebus	2300	9
		BRG-003	Indomie Goreng	2700	10
		BRG-004	Indomie Rebus	2500	5
KD-1002	01/05/2021	BRG-005	Mie Sedap Goreng	2600	7
		BRG-006	Mie Sedap Rebus	2400	8
KD-1003	02/05/2021	BRG-007	Pop Mie	5000	10

Contoh data pada Tabel 2 merupakan data yang belum dinormalisasi, selanjutnya menuju tahap normalisasi 1NF.

B. Membuat bentuk Normal ke 1 (1NF)

Suatu tabel dikatakan 1NF jika dan hanya jika setiap atribut dari data tersebut hanya memiliki nilai tunggal dalam satu baris. Jadi, tabel yang belum dinormalisasi tadi perlu diubah, sehingga bentuk 1NF, seperti terlihat pada Tabel 2.

Tabel 2. Bentuk Normal 1(1NF)

Kode Faktur	Tanggal	Kode Barang	Nama Barang	Harga Barang	QTY
KD-1001	01/05/2021	BRG-001	Supermie Goreng	2500	8
KD-1001	01/05/2021	BRG-002	Supermie Rebus	2300	9
KD-1001	01/05/2021	BRG-003	Indomie Goreng	2700	10
KD-1001	01/05/2021	BRG-004	Indomie Rebus	2500	5
KD-1002	01/05/2021	BRG-005	Mie Sedap Goreng	2600	7
KD-1002	01/05/2021	BRG-006	Mie Sedap Rebus	2400	8
KD-1003	02/05/2021	BRG-007	Pop Mie	5000	10

Inti dari normalisasi 1NF adalah tidak boleh ada *grouping* data ataupun duplikasi data. Sekarang lanjut pada tahap normalisasi 2NF.

C. Membuat ke Bentuk Normal ke-2 (2NF)

Syarat 2NF adalah tidak diperkenankan adanya *partial* "***functional dependency***" kepada *primary key* dalam sebuah tabel. Apa itu *"functional dependency"*?

Functional dependency adalah setiap atribut yang bukan kunci (*non key*) bergantung secara fungsional terhadap *primary key*.

Intinya adalah pada tahap normalisasi 2NF ini tabel tersebut harus dipecah berdasarkan *primary key*. Sehingga bentuk normalisasi 2NF dari tabel tersebut seperti pada Tabel 3 dan Tabel 4.

Hasil Normalisasi Bentuk Normal 2 (2NF)

Tabel 3. Tabel Barang

Kode Barang	Nama Barang	Harga
BRG-001	Supermie Goreng	2500
BRG-002	Supermie Rebus	2300
BRG-003	Indomie Goreng	2700
BRG-004	Indomie Rebus	2500
BRG-005	Mie Sedap Goreng	2600
BRG-006	Mie Sedap Rebus	2400
BRG-007	Pop Mie	5000

Tabel 4. Tabel Transaksi

Kode Faktur	Tanggal	Kode Barang	QTY
KD-1001	01/05/2021	BRG-001	8
KD-1001	01/05/2021	BRG-002	9
KD-1001	01/05/2021	BRG-003	10
KD-1001	01/05/2021	BRG-004	5
KD-1002	01/05/2021	BRG-005	7
KD-1002	01/05/2021	BRG-006	8
KD-1003	02/05/2021	BRG-007	10

Setelah dinormalisasi 2NF, tabelnya terpecah menjadi 2. Sekarang lanjut pada tahap normalisasi 3NF.

D. Bentuk Normal ke 3 (3NF)

Pada 3NF tidak diperkenankan adanya partial “transitive dependency” dalam sebuah tabel. Apa itu “transitive dependency”?

Transitive dependency biasanya terjadi pada tabel hasil relasi, atau kondisi dimana terdapat tiga atribut A, B, C. Kondisinya adalah A ⇒ B dan B ⇒ C. Maka C dikatakan sebagai *transitive dependency* terhadap A melalui B.

Intinya pada 3NF ini, jika terdapat suatu atribut yang tidak bergantung pada *primary key* tapi bergantung pada field yang lain maka atribut-atribut tersebut perlu dipisah ke tabel baru.

Contohnya ada pada atribut **qty**, kolom tersebut tidak bergantung langsung pada *primary key* **kode_faktur** melainkan bergantung pada kolom **kode_barang**. Jadi setelah dinormalisasi 3NF akan menghasilkan Tabel 5, Tabel 6 dan Tabel 7.

Tabel 5. Bentuk Normal ke-3 (3NF) untuk Tabel Barang

Kode Barang	Nama Barang	Harga
BRG-001	Supermie Goreng	2500
BRG-002	Supermie Rebus	2300
BRG-003	Indomie Goreng	2700
BRG-004	Indomie Rebus	2500
BRG-005	Mie Sedap Goreng	2600
BRG-006	Mie Sedap Rebus	2400
BRG-007	Pop Mie	5000

Tabel 6. Bentuk Normal ke-3 (3NF) untuk Tabel Transaksi

Kode Faktur	Tanggal
KD-1001	01/05/2021
KD-1002	01/05/2021
KD-1003	02/05/2021

Tabel 7. Bentuk Normal ke-3 (3NF) untuk Tabel Detail Barang

Kode Faktur	Kode Barang	QTY	Harga
KD-1001	BRG-001	8	2500
KD-1001	BRG-002	9	2300
KD-1001	BRG-003	10	2700
KD-1001	BRG-004	5	2500
KD-1002	BRG-005	7	2600
KD-1002	BRG-006	8	2400
KD-1003	BRG-007	10	5000

Hasil Normalisasi 3NF dapat dilihat pada tahap normalisasi 3NF menghasilkan 1 tabel baru dari hasil pemecahan **tabel transaksi** (Tabel 6)

yaitu **tabel detail barang** (Tabel 7) yang isinya menampung barang-barang yang dibeli.

Perhatikan Tabel 7, kenapa di tabel detail barang terdapat kolom harga lagi? padahal kolom harga sudah ada di tabel barang. Kolom harga pada tabel detail barang digunakan untuk menyimpan harga barang pada saat proses transaksi. Jadi, meskipun kolom harga pada tabel barang berubah (naik/turun), harga barang yang ada pada tabel detail barang tidak ikut berubah (fixed). Bayangkan jika kita tidak menambahkan kolom harga pada pada tabel detail barang, maka yang terjadi total invoice dari transaksi akan berubah seiring berubahnya harga barang.

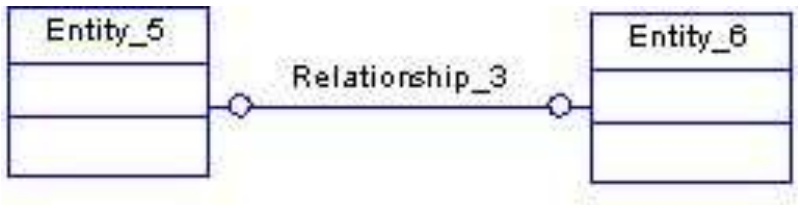
BAB 3. RELASI TABEL DATABASE

Relasi adalah hubungan antara tabel yang mempresentasikan hubungan antar objek di dunia nyata. Relasi merupakan hubungan yang terjadi pada suatu tabel dengan lainnya yang mempresentasikan hubungan antar objek di dunia nyata dan berfungsi untuk mengatur mengatur operasi suatu database. Hubungan yang dapat dibentuk dapat mencakup 3 macam hubungan, yaitu: *One to One, One to Many, dan Many to Many*. (Purwoko & Sulaiman, 2021)

Relasi *One to One, One to Many,* dan *Many to Many* pada sebuah tabel database akan sering kita lakukan disaat merancang sebuah database yang baik. Sebelum lebih lanjut lagi, relasi tabel pada database itu artinya apa sih? Relasi pada tabel merupakan relasi atau hubungan antara tabel yang satu dengan yang lain pada database. (Yuliansyah et al., 2014)

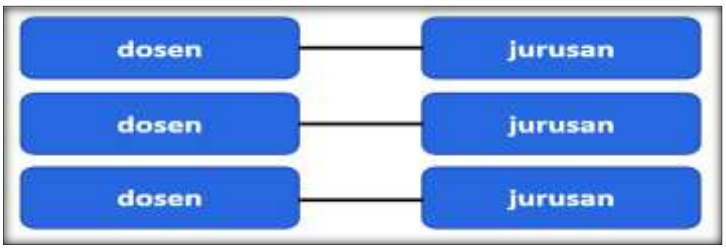
3.1 One-To-One (1-1)

Mempunyai pengertian “Setiap baris data pada tabel pertama dihubungkan hanya ke satu baris data pada tabel ke dua”, ilustrasi seperti pada Gambar 10. Contohnya : relasi antara tabel mahasiswa dan tabel orang tua. Satu baris mahasiswa hanya berhubungan dengan satu baris orang tua begitu juga sebaliknya.



Gambar 10. Ilustrasi Relasi Tabel *One to One*

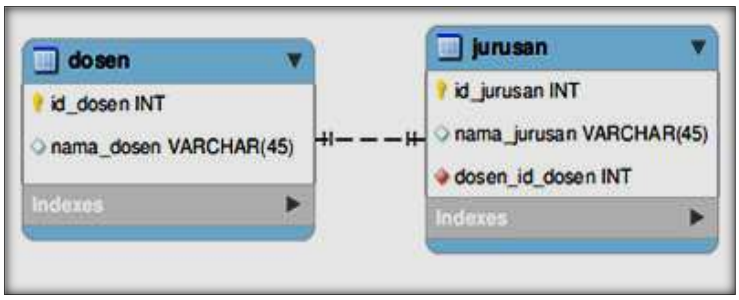
Relasi *One to One* adalah relasi yang mana setiap satu baris data pada tabel pertama hanya berhubungan dengan satu baris pada tabel kedua. Agar tidak bingung, lihat visualisasi garis relasi *One to One* seperti pada Gambar 11.



Gambar 11. Visualisasi relasi *One to One*

Pada Gambar 11 maksudnya adalah satu jurusan dikepalai oleh satu dosen.

Contoh tabel relasi *One to One* seperti pada Gambar 12.

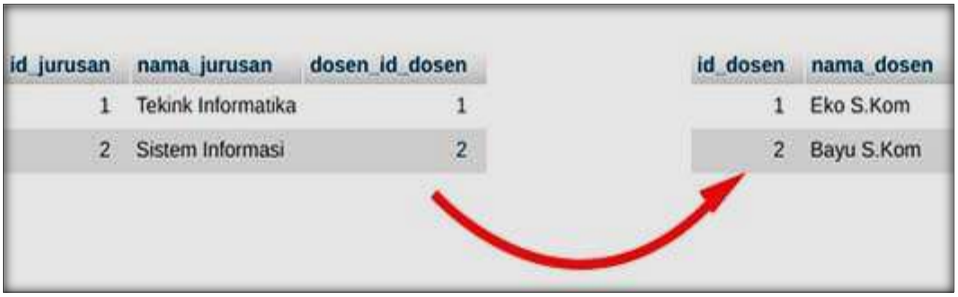


Gambar 12. Contoh Relasi tabel *One to One*

Pada tabel jurusan terdapat *primary key* id_jurusan dan *foreign key* **dosen_id_dosen**. Dimana *foreign key* itulah yang digunakan sebagai penghubung tabel dosen.

Contoh tabel relasi *One to One* dengan **beberapa isi data.**

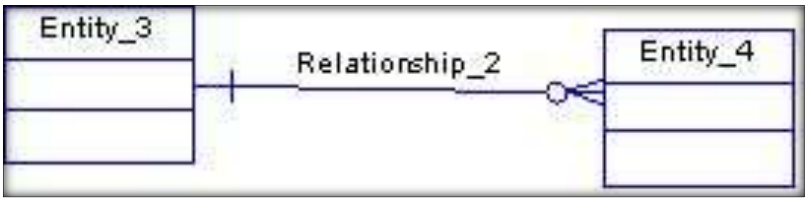
Jika masih bingung dengan gambaran Gambar 12, Gambar 13 merupakan contoh jika tabel tersebut diisi beberapa data.



Gambar 13. Contoh isi data pada tabel *one to one*

3.2 *One-To-Many* (1-N)

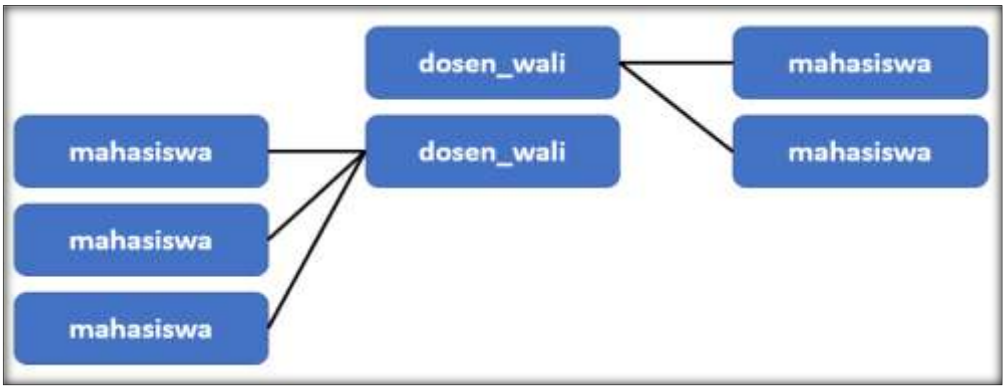
Mempunyai pengertian “Setiap baris data dari tabel pertama dapat dihubungkan ke satu baris atau lebih data pada tabel ke dua”, ilustrasi seperti pada Gambar 14. Contohnya : relasi perwalian antara tabel dosen dan tabel mahasiswa. Satu baris dosen atau satu dosen bisa berhubungan dengan satu baris atau lebih mahasiswa.



Gambar 14. Ilustrasi relasi *One to Many*

contoh relasi *one-to-many*

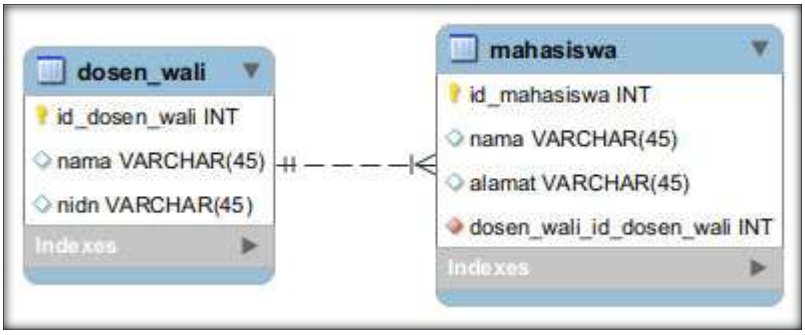
Relasi *One to Many* adalah relasi yang mana setiap satu baris data pada tabel pertama berhubungan dengan lebih dari satu baris pada tabel kedua. Agar tidak bingung, lihat visualisasi garis relasi *One to Many* seperti Gambar 15.



Gambar 15. Visualisasi relasi One to Many

Pada Gambar 15 maksudnya adalah satu dosen wali dapat menampung lebih dari satu mahasiswa.

Contoh tabel relasi *One to Many* seperti pada Gambar 16 .



Gambar 16. Relasi tabel *One to Many*

Pada tabel mahasiswa terdapat *primary key* **id_mahasiswa** dan *foreign key* **dosen_wali_id_dosen_wali**. Yang mana *foreign key* itulah yang digunakan sebagai penghubung tabel dosen_wali.

Contoh tabel relasi *One to Many* **dengan beberapa isi data.**

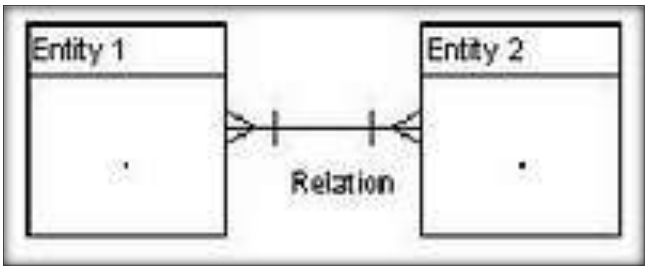
Jika masih bingung dengan gambaran tabel di atas, Gambar 17 merupakan contoh jika tabel tersebut diisi beberapa data.

Gambar 17. Contoh tabel relasi One to Many dengan beberapa isi data

3.3 *Many-To-Many* (N-M)

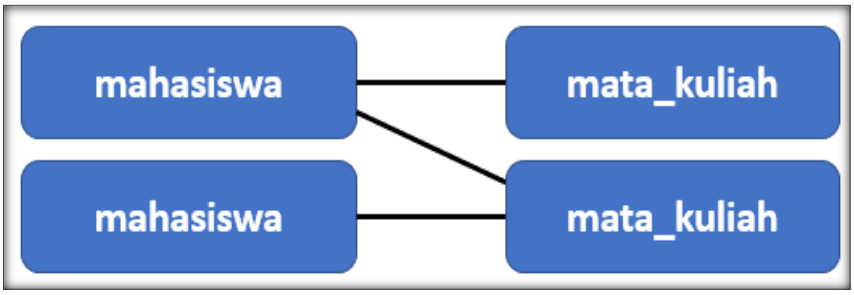
Mempunyai pengertian “Satu baris atau lebih data pada tabel pertama bisa dihubugkan ke satu atau lebih baris data pada tabel ke dua”, ilustrasi seperti Gambar 18. Artinya ada banyak baris di tabel satu dan tabel dua yang saling berhubungan satu sama lain. Contohnya : relasi antar tabel mahasiswa dan tabel

mata kuliah. Satu baris mahasiswa bisa berhubungan dengan banyak baris mata kuliah begitu juga sebaliknya.



Gambar 18. Ilustrasi relasi *Many-To-Many*

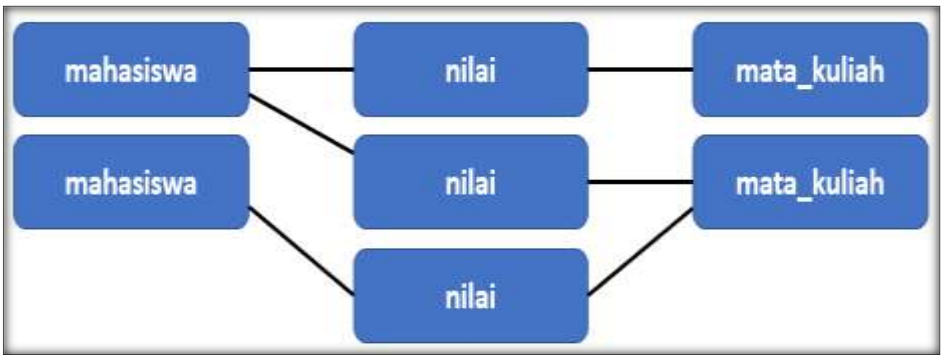
Relasi *Many to Many* adalah relasi yang mana setiap lebih dari satu baris data dari tabel pertama berhubungan dengan lebih dari satu baris data pada tabel kedua. Artinya, kedua tabel masing-masing dapat mengakses banyak data dari tabel yang direlasikan. Dalam hal ini, relasi *Many to Many* akan menghasilkan tabel ketiga sebagai perantara tabel kesatu dan tabel kedua sebagai tempat untuk menyimpan *foreign key* dari masing-masing tabel. Agar tidak bingung, lihat visualisasi garis relasi *Many to Many* seperti pada Gambar 19.



Gambar 19. Visualisasi relasi *Many to Many*

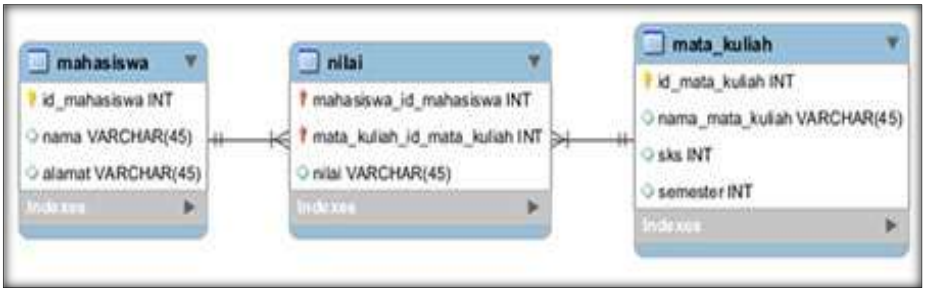
Pada Gambar 19 maksudnya adalah setiap mahasiswa dapat mengambil banyak mata kuliah dan setiap mata kuliah dapat diambil banyak mahasiswa.

Karena relasi One to Many menghasilkan tabel baru atau tabel ketiga, jika menyertakan *record* tabel baru tersebut pada grafik akan terlihat seperti di bawah ini. Karena hubungan erat mahasiswa yang belajar mata kuliah adalah nilai dan sekaligus berfungsi sebagai penghubung antara tabel mahasiswa dan mata_kuliah. Visualisasi relasi *Many to Many* seperti pada Gambar 20.



Gambar 20. Visualisasi relasi *Many to Many*

Gambar 21 merupakan **Contoh tabel relasi** *Many to Many*.



Gambar 21. Relasi tabel *Many to Many*

Coba perhatikan pada Gambar 21, terdapat tiga tabel yaitu tabel mahasiswa, nilai, dan mata_kuliah. Tabel mahasiswa dan mata_kuliah tersebut masing-masing berelasi *Many to Many* dan menghasilkan tabel baru yaitu tabel nilai. Sedangkan tabel baru atau tabel nilai tersebut sebagai penghubung antara tabel mahasiswa dan mata_kuliah yang mana tabel baru tersebut terdapat *foreign key* **mahasiswa_id_mahasiswa** dan **mata_kuliah_id_mata_kuliah** yang fungsinya untuk mengakses tabel mahasiswa dan mata_kuliah.

Contoh tabel relasi *Many to Many* **dengan beberapa isi data.**

Jika masih bingung dengan gambaran tabel pada Gambar 21, maka Gambar 22 merupakan contoh jika tabel tersebut diisi beberapa data.

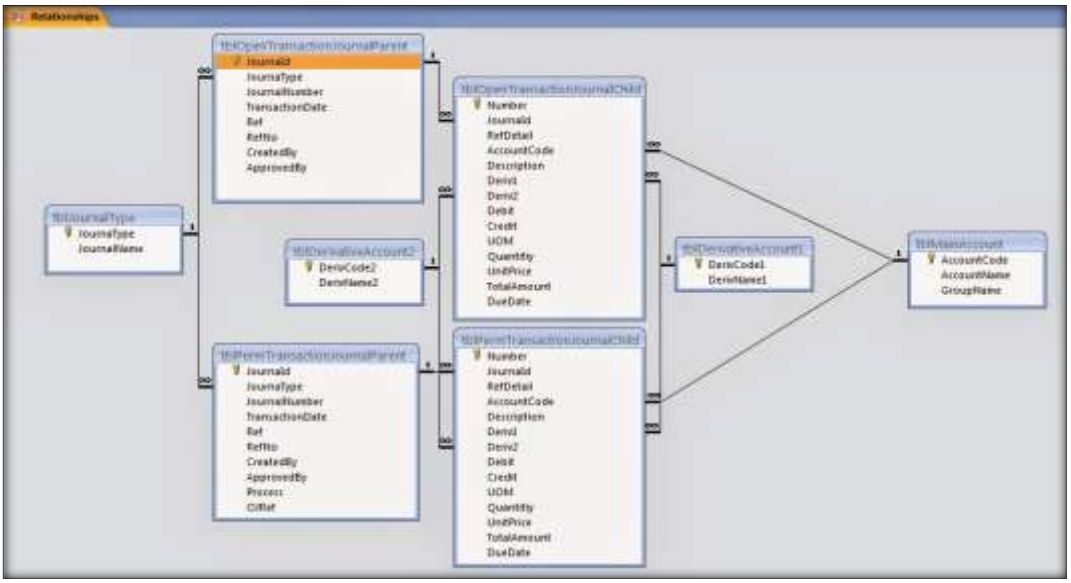
mahasiswa_id_mahasiswa	mata_kuliah_id_mata_kuliah	nilai
1	2	90
2	1	95

id_mahasiswa	nama	alamat
1	Zainuri	Nganjuk
2	Kurnia	Surabaya

id_mata_kuliah	nama_mata_kuliah	sks	semester
1	Pemrograman Mobile	4	5
2	Kecerdasan Buatan	2	5

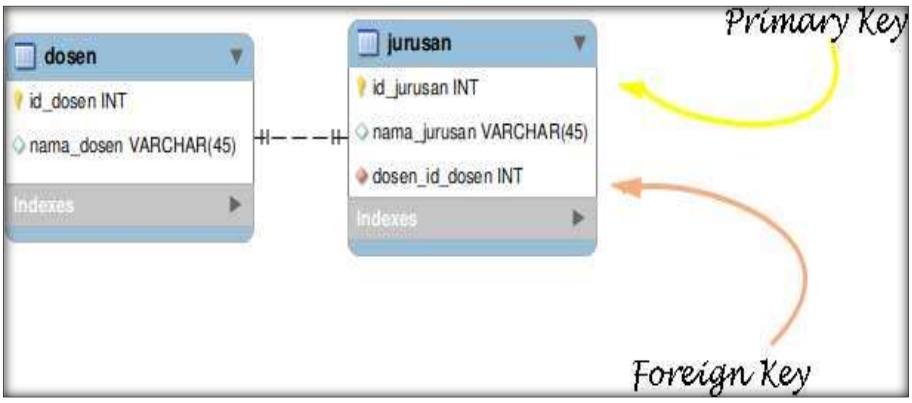
Gambar 22. Contoh tabel relasi One to Many dengan beberapa isi data.

Gambar 23 merupakan contoh Relasi Tabel



Gambar 23. Contoh Relasi Tabel

Pada sebuah database, relasi dihubungkan dengan dua tabel yang dihubungkan melalui kolom *foreign key* pada tabel pertama dengan *primary key* tabel kedua. Masih bingung? coba amati Gambar 24.



Gambar 24. *Primary Key* dan *Foreign Key*

Berdasarkan Gambar 24 terdapat 2 tabel yaitu tabel dosen dan tabel jurusan yang memiliki *primary key* dan *foreign key*. Pada tabel jurusan terdapat *primary key* **id_jurusan** dan *foreign key* **dosen_id_dosen** yang berelasi ke tabel dosen. Sedangkan isi dari *foreign key* **dosen_id_dosen** adalah **id_dosen** pada tabel dosen. *Foreign key* yang ada pada tabel jurusan digunakan untuk menghubungkan tabel dosen dengan tabel jurusan.

Beberapa hal yang harus kamu ketahui tentang *foreign key* & *primary key*:

- Harus unik
- Tabel hanya boleh memiliki satu *primary key*, namun dalam beberapa kasus boleh lebih dari 1 *primary key* (*composite key*)
- Tabel boleh memiliki lebih dari satu *foreign key*
- *Foreign key* digunakan untuk membuat relasi antar tabel

BAB 4. *ENTITY RELATIONSHIP DIAGRAM* (ERD)

Entity Pelationship Diagram (ERD) adalah suatu model informasi untuk menjelaskan suatu hubungan antara data dan basis data yang digambarkan dengan sebah grafik dan juga notasi dengan model data konseptual.

Loonam dan *Brady* yaitu seorang para ahli bahasa mengatakan bahwa *Entity Pelationship Diagram (*ERD) merupaka sebuah teknik yang dipakai untuk menjelaskan data yang dibutuhkan dalam sebuah organisasi. Permodelannya bisa seperti sistem analisis yaitu tahapan analisa persyaratan dalam proyek pengembangan sistem. (Al-Masree, 2015)

4.1 Komponen ERD

Komponen ERD (*Entity Pelationship Diagram*) terdiri dari:

A. Entitas

Entitas dalam ERD merupakan sebuah objek atau simbol yang berfungsi sebagai identitas pada kesatuan yang mempunyai nama dan label. Entitas digambarkan dengan simbol/objek sebuah persegi panjang.

B. Relasi (Hubungan Antar Entitas)

Relasi dalam ERD merupakan sebuah objek atau simbol yang menghubungkan antara satu entitas atau lebih yang tidak memiliki fisik namun ia hanya sebagai konseptual, relasi juga berfungsi untuk mengetahui jenis hubungan antara 2 data. Relasi digambarkan dengan simbol/bentuk belah ketupat. **Derajat relasi atau kardinalitas rasio**, yiatu jumlah maksimum relasi antara entitas dengan entitas lainnya. Bentuk Relasi:

- ***One to One*** **(1:1)**, yaitu setiap satu anggota entitas hanya boleh ber-relasi dengan satu anggaota entitas lain dan sebaliknya.

- ***One to many* (1:M / Many)**, yaitu setiap satu anggota entitas boleh ber-relasi dengan anggota entitas lain lebih dari satu.
- ***Many to Many* (M:M)**, yaitu setiap satu anggota entitas ber-relasi dengan banyak himpunan anggota entitas lain.

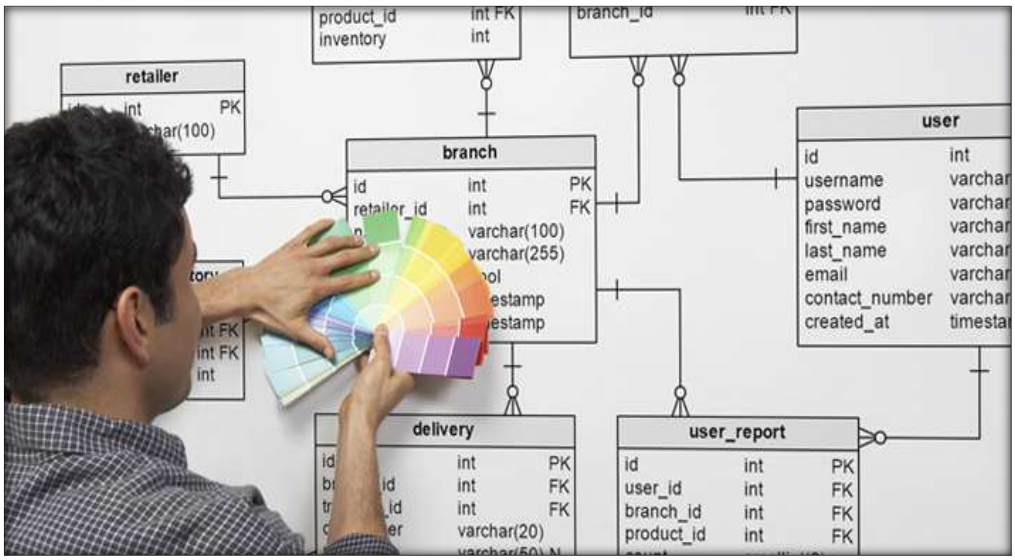
C. Atribut

Atribut dalam ERD merupakan sebuah objek atau simbol sebagai karakteristik dari entitas atau relasi yang menampilkan penjelasan informasi detail tentang keduanya. Atribut ini juga digambarkan dengan simbol/bentuk elips, dan memiliki beberapa fungsi yaitu:

- ***Attribute Key*** (atribut yang memiliki satu atau gabungan dengan atribut lain, dalam tabel unik).
- ***Attribute Simple*** (atribut yang bernilai atomik, yaitu tidak bisa dipecah-pecah lagi).
- ***Attribute Multivalue*** (atribut yang memiliki lebih dari satu nilai / miltivalue).
- ***Attribute Composite*** (atribut dengan bentuk oval yang lebh kecil dari atribut lain sebagai sub-atribut).
- ***Attribute Derrivatif*** (atribut dengan bentuk oval dengan garis putus-putus, yaitu sebuah hasil dari atribut lain atau dari relasi).

D. Alur

Alur dalam ERD merupakan sebuah objek/simbol yang berfungsi sebagai simbol penghubung antara atribut dan entitas dan hubungan antara entitas dan relasi. Alur disimbolkan dengan bentuk garis.



Gambar 25. Alur ERD

Fungsi, Manfaat atau Peran dari ERD (*Entity Relationship Diagram*)

1. ERD untuk memodelkan struktur data dan hubungan antar data, yang digambarkan menggunakan beberapa notasi dan symbol.
2. Menghilangkan redundansi data.
3. mampekecil jumlah relasi di dalam basis data.
4. Menjadikan relasi normal, sehingga dapat memperkecil permasalahan pada pembaharuan, penambahan dan penghapusan.
5. Model untuk struktur data dan hubungan antar data.
6. Menentukan Entitas yaitu konsep, kejadian, peran, lokasi dan hal nyata untuk penggunaan menyimpan data .
7. Menentukan hubungan antar pasangan entitas menggunakan matriks relasi.
8. Entitas digambarkan dengan kotak dan relasi digambarkan dengan garis.
9. Menentukan jumlah kejadian satu entitas dalam kejadian entitas yang berhubungan.
10. Menentukan atribut yaitu menentukan lapangan yang di perlukan system
11. Bisa diuji dengan mengabaikan proses.
12. Menjelaskan hubungan antara data dalam basis data, dengan objek-objek dasar data yang memiliki hubungan antar relasi.

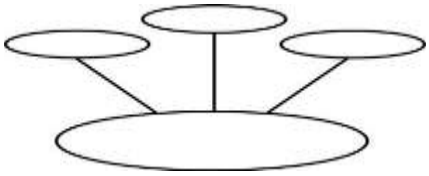
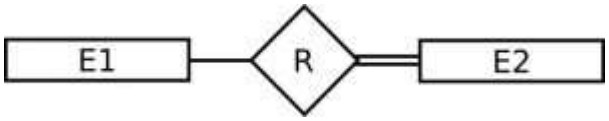
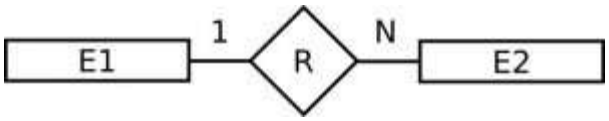
13. m*end*okumentasikan data-data yaitu dengan mengidentifikasi jenis entitas dan relasinya.
14. Memudahkan kita untuk menganalisi dan mengetahui perubahan sistem dari awal.
15. Gambaran umum untuk sistem yang akan dibuat, dan dapat memudahkan para developer.
16. Menghasilkan dokumentasi yang baik untuk *disk*usi.
17. Representasi visual, yaitu menawarkan persentasi visual dari tata letak.
18. Menjadikan sebuah komunikasi yang efektif.
19. Fleksibilitas tinggi.

4.2 Simbol ERD (Entity Pelationship Diagram)

Ada beberapa simbol yang dignakan, yang sebelumnya sudah disinggung di atas, Tabel 8 merupakan simbol-simbol ERD:

Tabel 8. Simbol ERD

No	Nama	Simbol
1.	**Entity**	
2.	**Weak Entity**	
3.	**Relationship**	
4.	**Identifying Relationship**	
5.	**Atibut**	
6.	**Atribut Key**	
7.	**Atribut Multivalue**	

No	Nama	Simbol
8.	**Atribut Derivatif**	
9.	**Atribut Kompusit**	
10.	**Total Participation of E1 in R**	
11.	**Cardinality Ratio 1:N E1:E2 in R**	

4.3 Cara membuat ERD (*Entity Pelationship Diagram*)

Dalam pembuatan ERD ada dua cara yaitu secara manual menggunakan kertas dan alat tulis, dan dengan digital. Kita bisa membuatnya ERD secara digital dengan menggunakan software khusus yaitu Microsoft Visio baik yang 2013, 2014, 2015, 2016, atau yang tebaru.

Berikut langkah-langkah dalam membuat sebuah ERD:

1. Menentukan Entitas.
2. Menentikan Relasi.
3. Membuat gambar ERD sementara.
4. Mengisi kardinalitas.
5. Menentukan kunci utama (*primary key*).
6. Menggambar ERD menurut kunci utama.
7. Menentukan Atribut.
8. Pemetaan Atribut.
9. Menggambar ERD dengan Atribut.
10. Memeriksa hasil.

BAB 5. *SQL SERVER MANAGEMENT STUDIO*

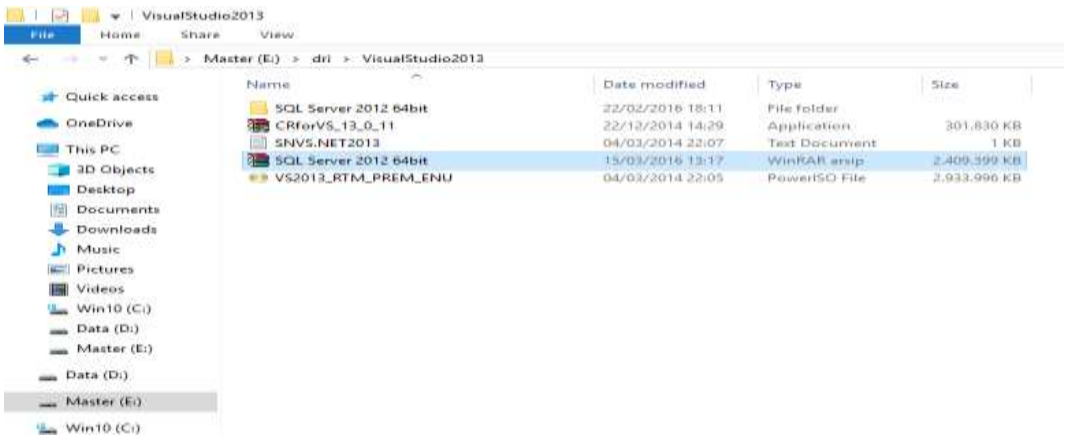
Pada dasarnya SQL Server itu sendiri adalah bahasa yang dipergunakan untuk mengakses data dalam basis data *relation*. Bahasa ini secara *defacto* adalah bahasa standar yang digunakan dalam manajemen basis data relasional. Saat ini hampir semua server basis data yang ada m*end*ukung bahasa ini dalam manajemen datanya.

SQL Server Management Studio atau yang biasa disingkat SSMS ini merupakan ruang lingkup untuk mengatur segala insfrastruktur SQL, dari SQL server ke SQL database. *SQL Server Management Studio* menyediakan alat / *tools* untuk mengkonfigurasi, memantau, dan mengelola instansi SQL. Kamu dapat menggunakan SSMS untuk *deploy*, monitoring, dan *upgrade* komponen *data-tier* yang digunakan pada aplikasi (termasuk membuat *query* dan *script*). (Dewson, 2015)

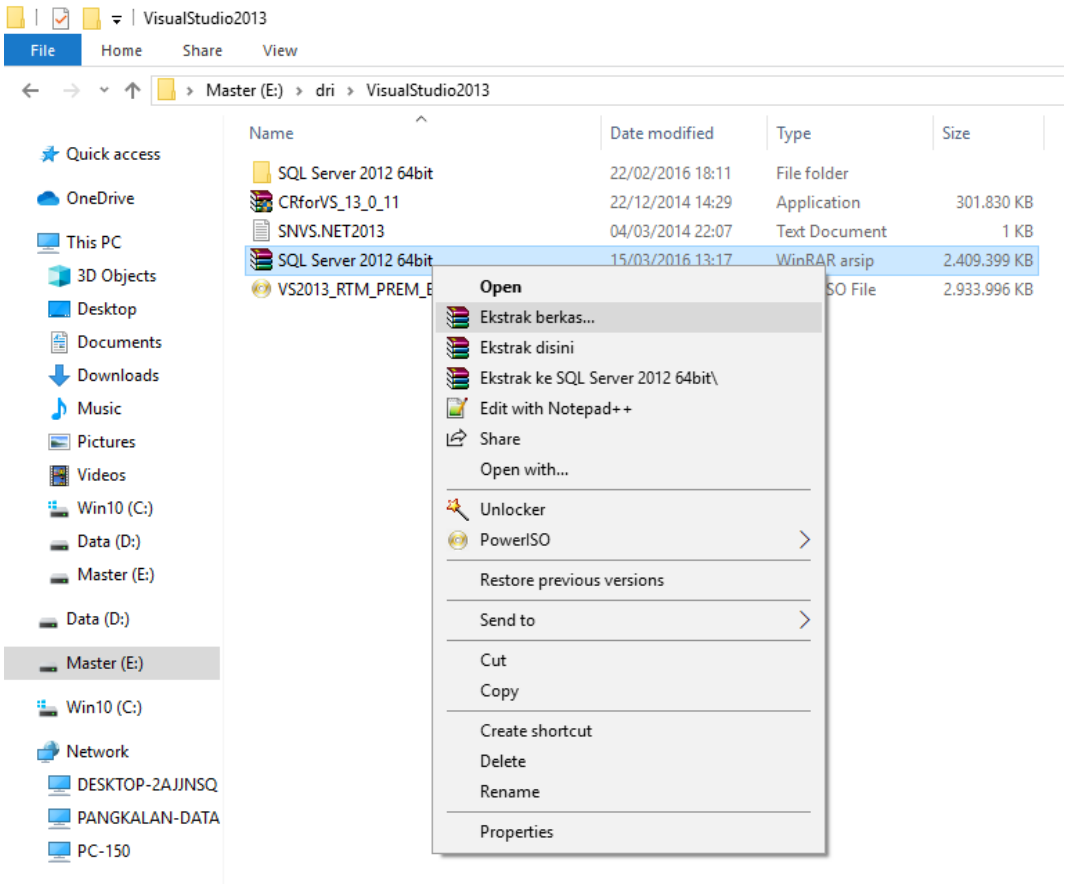
5.1 Cara Instalasi Sql Server Managemen Studio

Langkah-langkah untuk menginstal SQL Server 2012 *Management Studio Express* sebagai berikut:

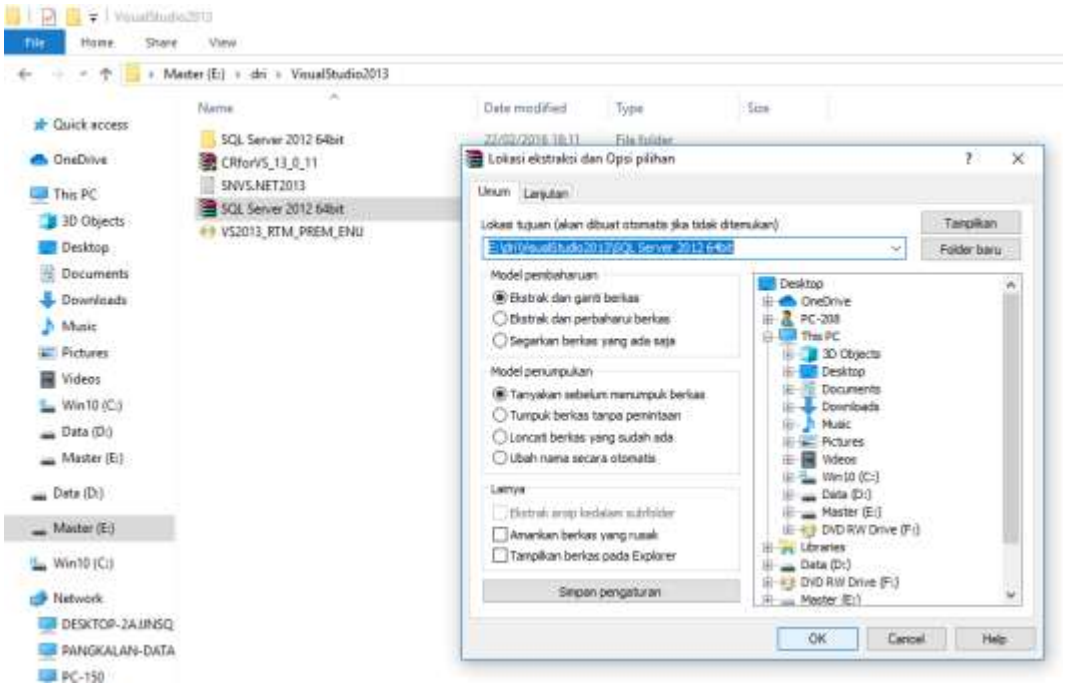
1) *Download* SQL Server 2012 *Management Studio Express* SP1 dari *Microsoft Download Center*. Pastikan untuk memilih file yang dirancang untuk sistem anda (x86 atau x64). Dalam contoh ini, saya akan men-*download* file "SQLManagementStudio_x86_ENU.exe" untuk sistem 64-**bit**.



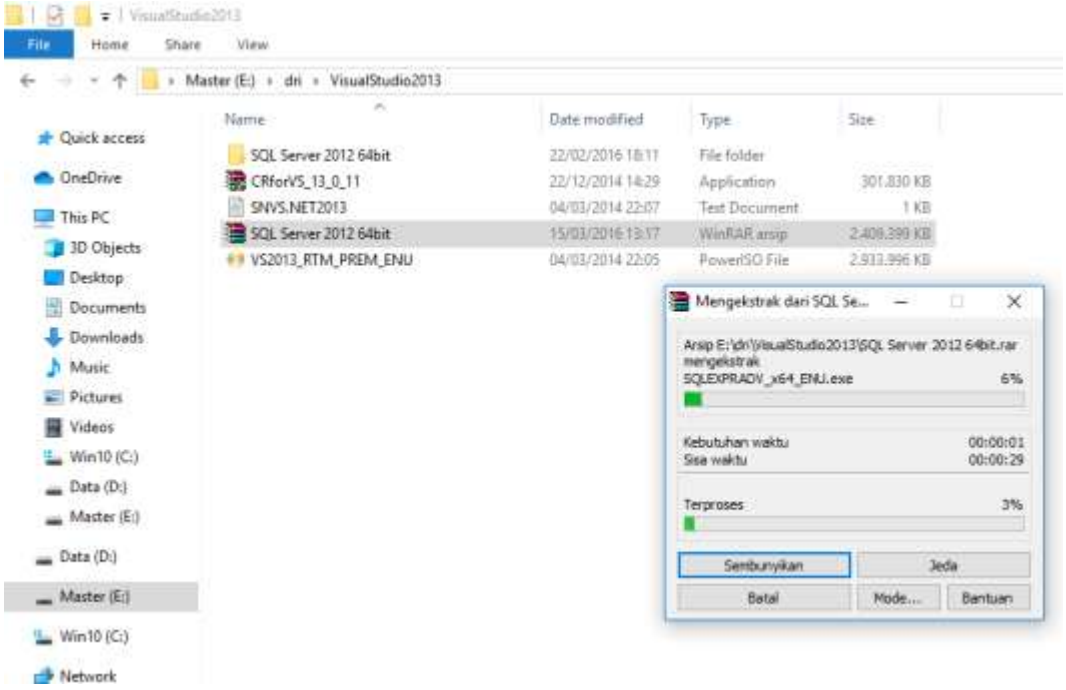
Gambar 26. SQLManagementStudio_x86_ENU.exe



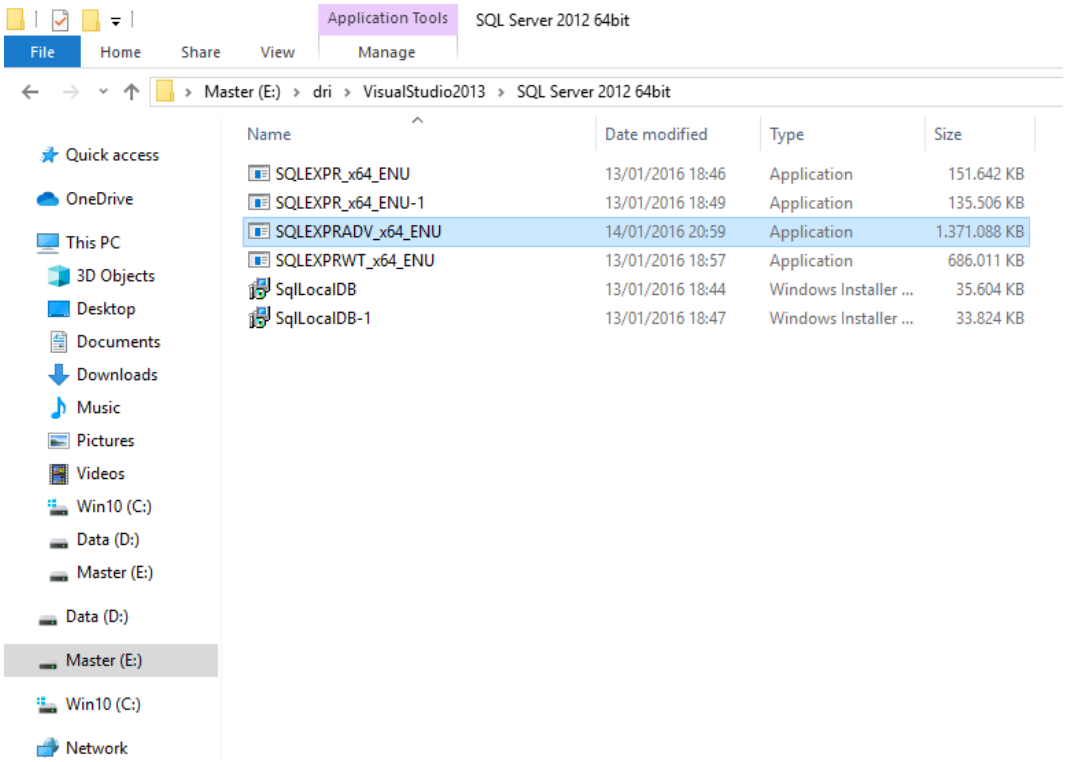
Gambar 27. Ekstrak SQLManagementStudio_x86_ENU.exe



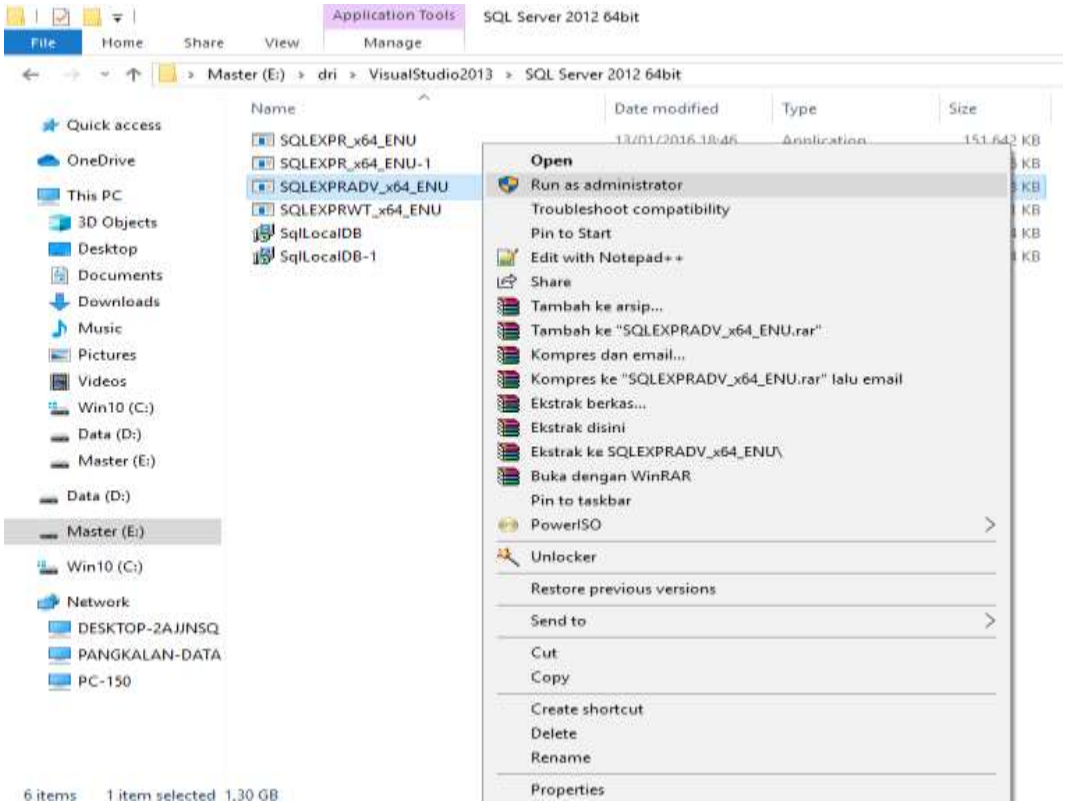
Gambar 28. Folder Ekstrak untuk SQLManagementStudio_x86_ENU.exe



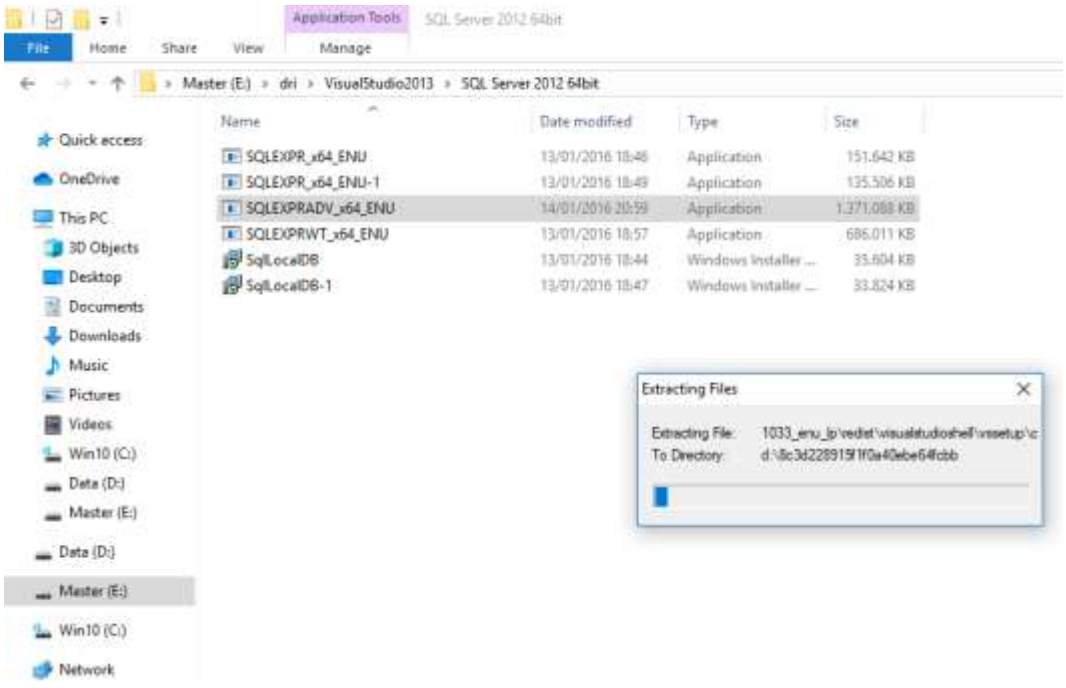
Gambar 29. Proses Ekstrak SQLManagementStudio_x86_ENU.exe



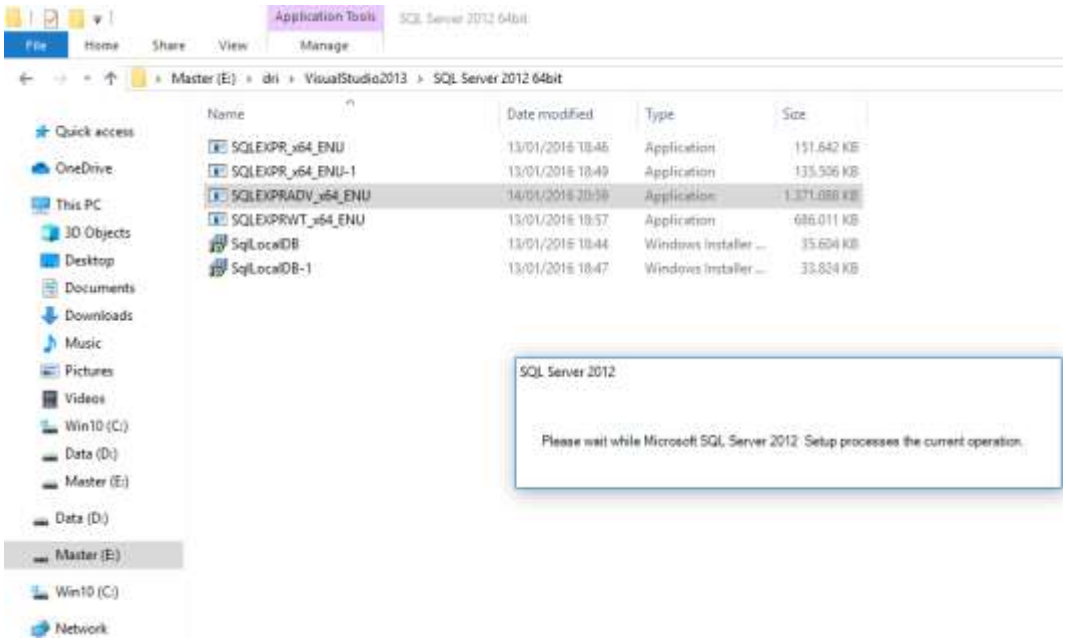
Gambar 30. Hasil Ekstrak SQLManagementStudio_x86_ENU.exe



Gambar 31. Menjalankan file SQLEXPADV_x64_ENU



Gambar 32. Proses Running SQLEXPADV_x64_ENU

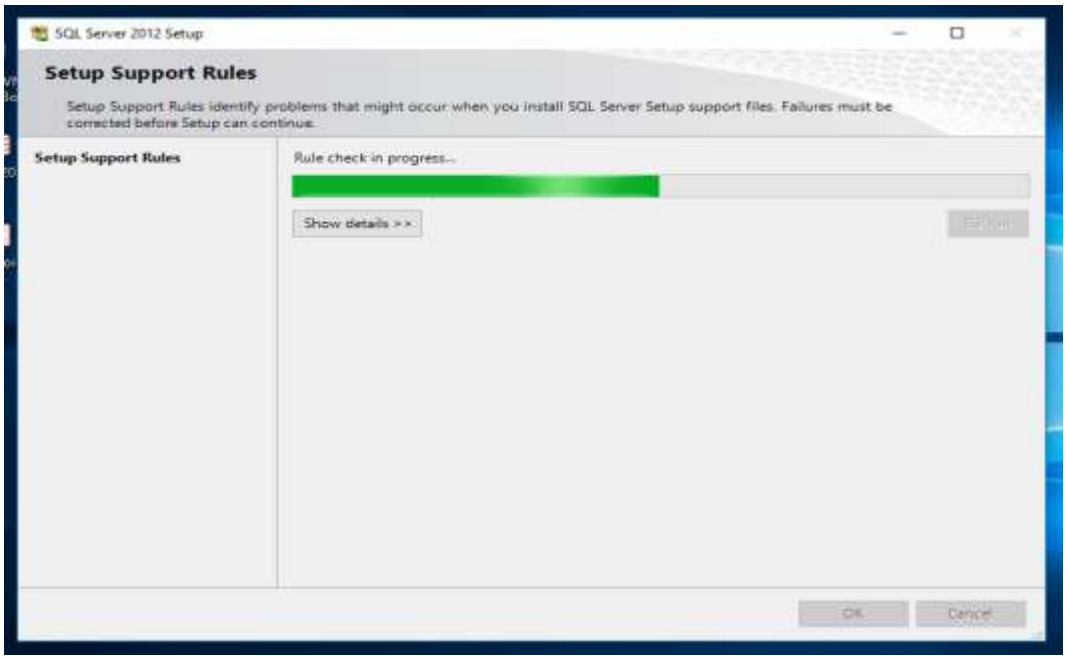


Gambar 33. Proses Running SQLEXPADV_x64_ENU sampai selesai

2) Jalankan file *setup*. Pilih Install → *New SQL Server stand-alone installation or add feature to an existing installation*.

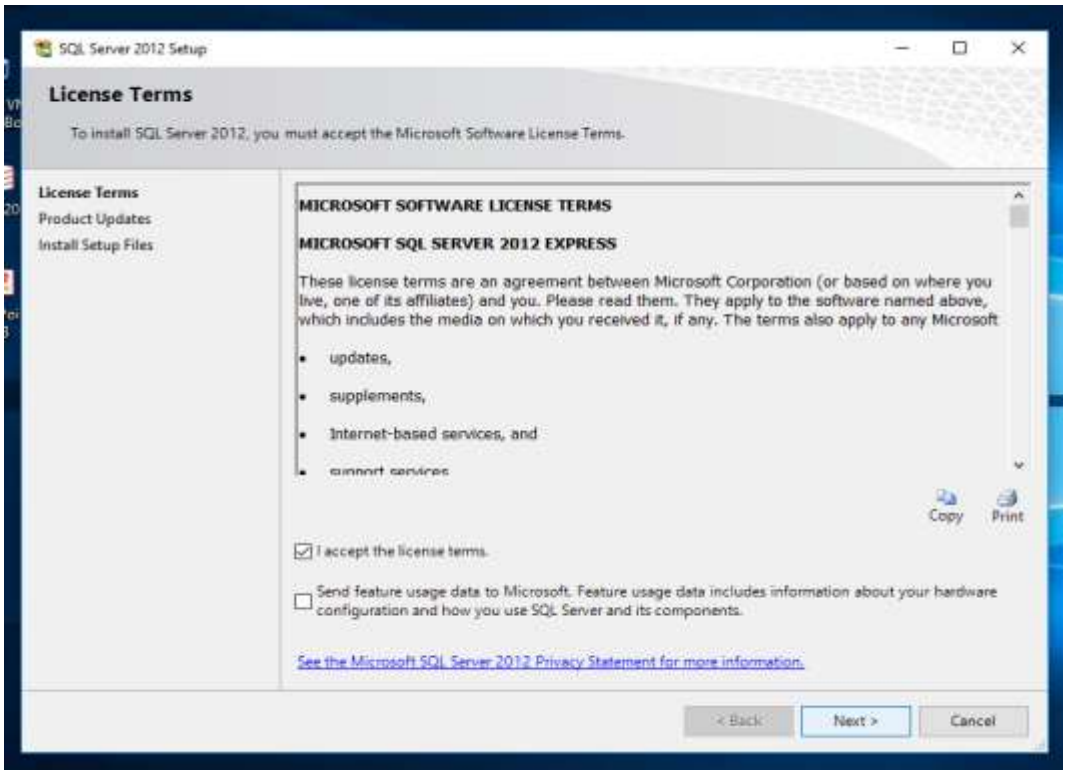
Gambar 34. *Install SQL Server*

3) Jalankan file *setup*. Pilih Install → *New SQL Server stand-alone installation or add feature to an existing installation*, selanjutnya akan muncul *Setup Support Rules*



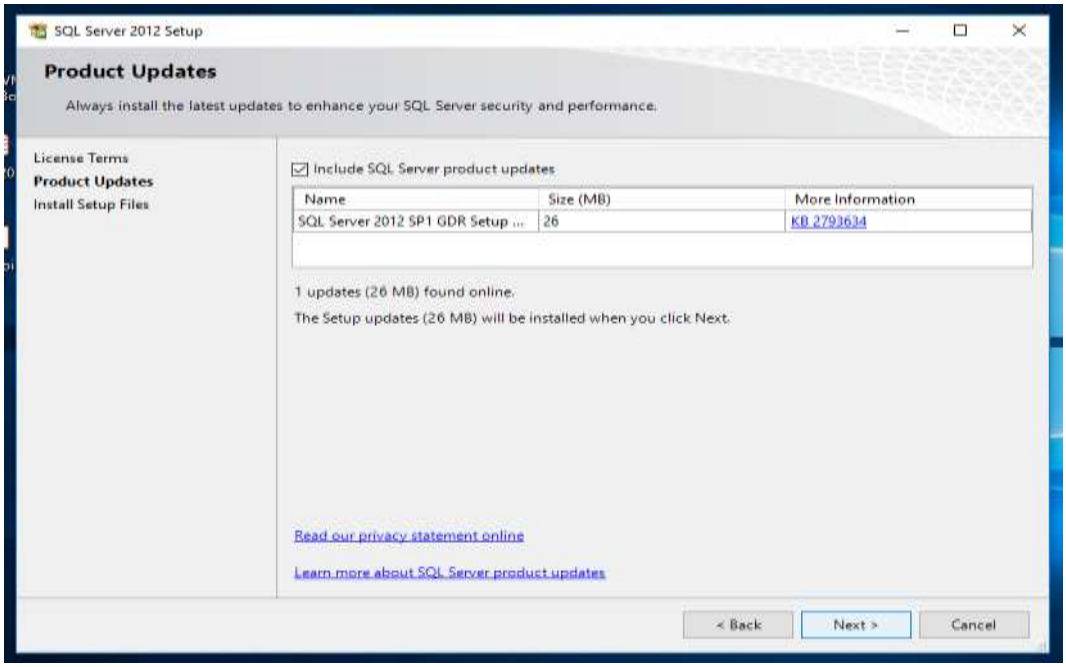
Gambar 35. *Setup Support Rules*

4) Langkah selanjutnya adalah License Terms → klik next



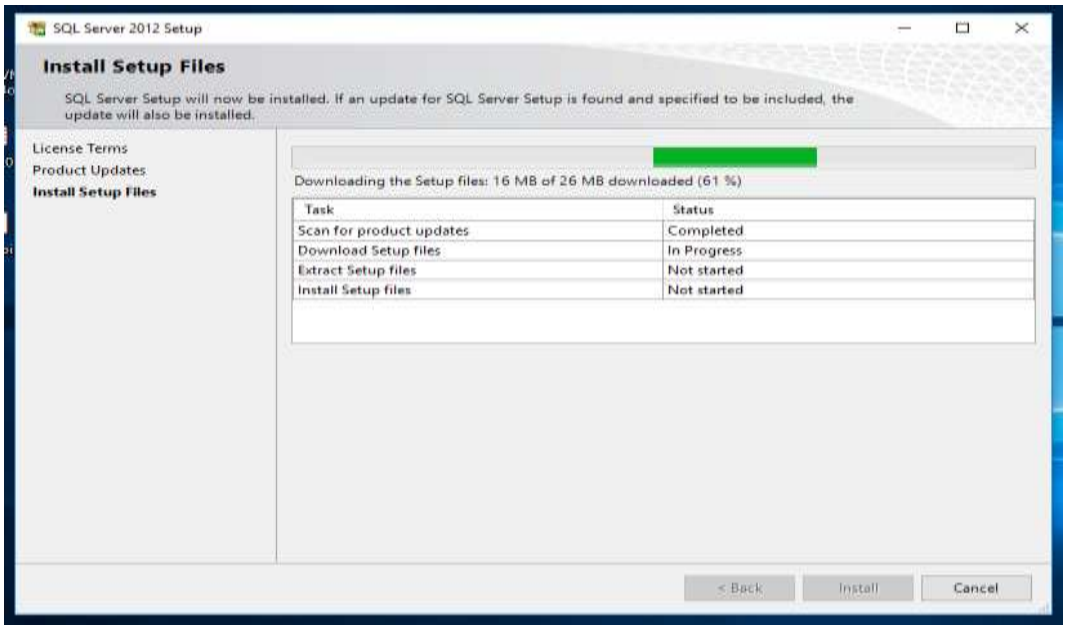
Gambar 36. *License Terms*

5) Langkah selanjutnya adalah pilih Product Update → klik next



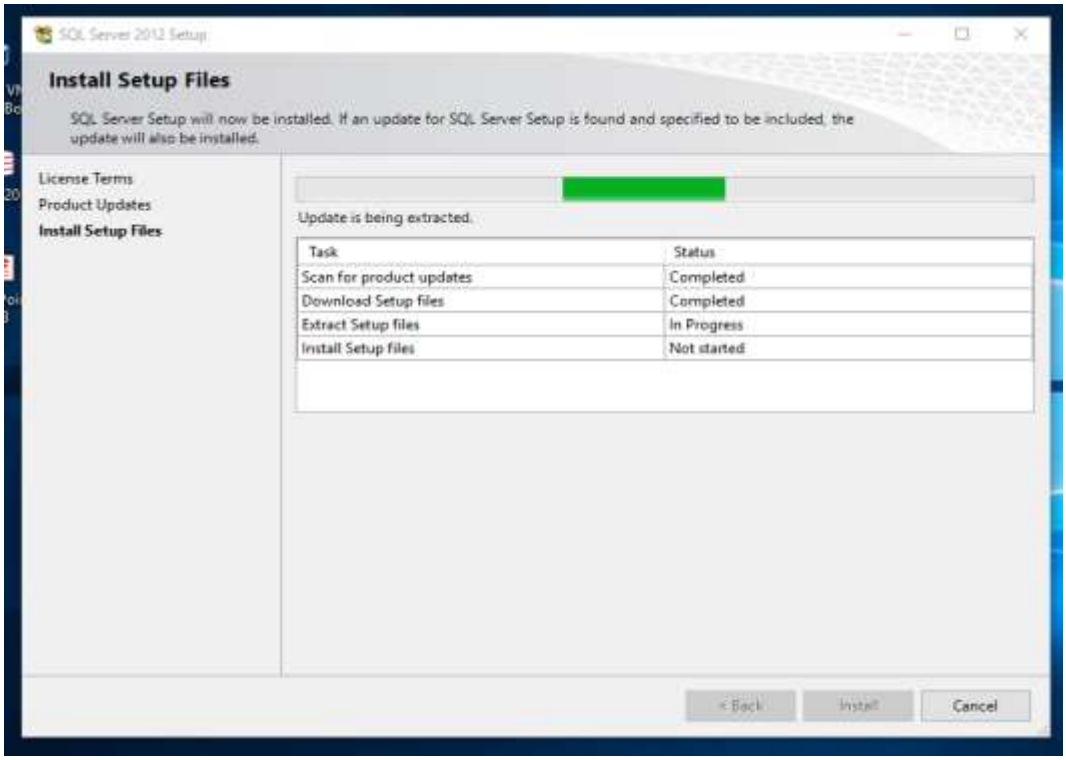
Gambar 37. *Product Update*

6) Langkah berikutnya pilih *Install Setup Files* tunggu beberapa saat



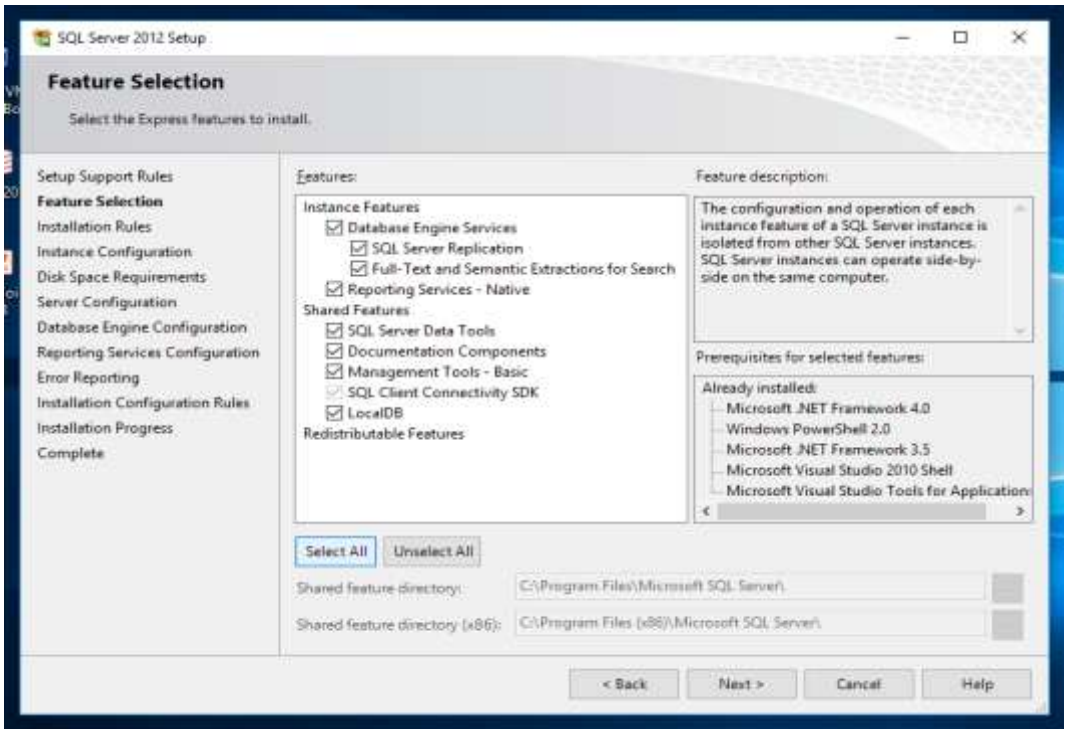
Gambar 38. *Install Setup Files*

7) Proses *Install Setup Files* masih berjalan tunggu sampai selesai



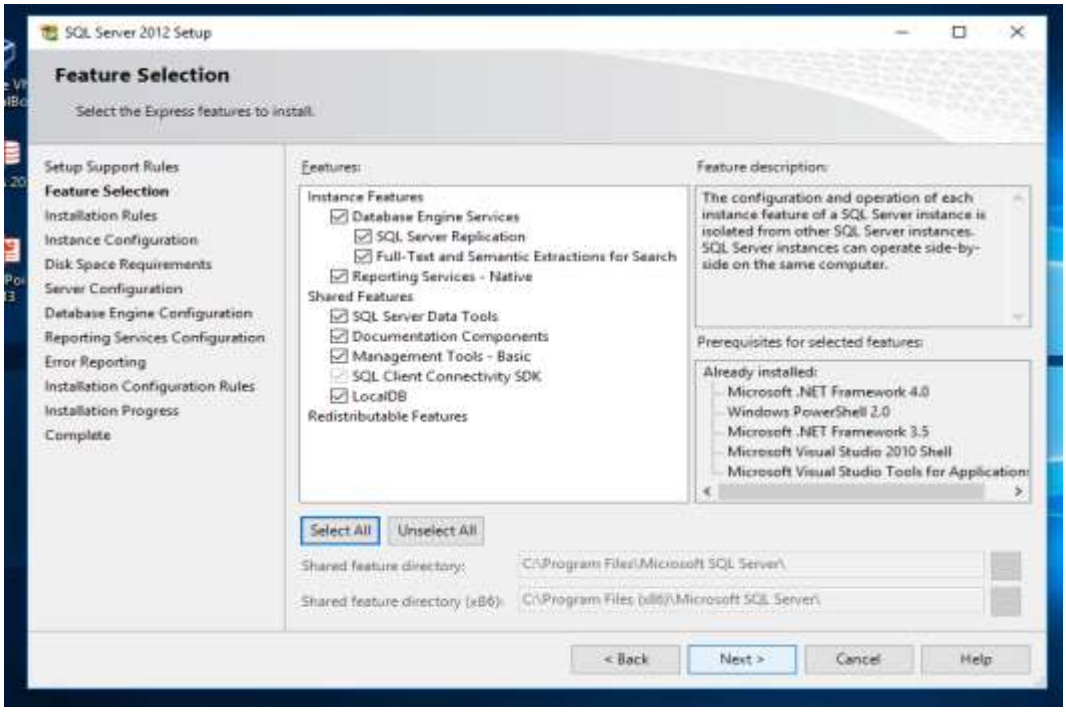
Gambar 39. Proses *Install Setup Files*

8) Langkah selanjutnya adalah *Feature Selection*→ klik *select all*



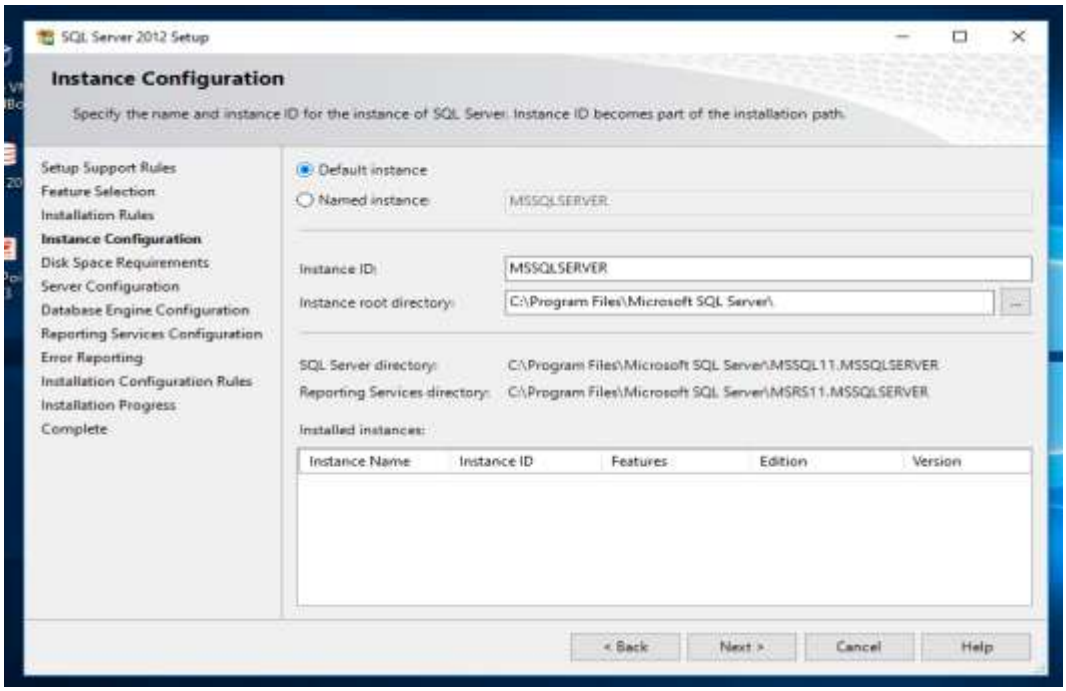
Gambar 40. *Feature Selection*

9) Langkah selanjutnya adalah *Feature Selection*→ klik *select all* →*klik next*



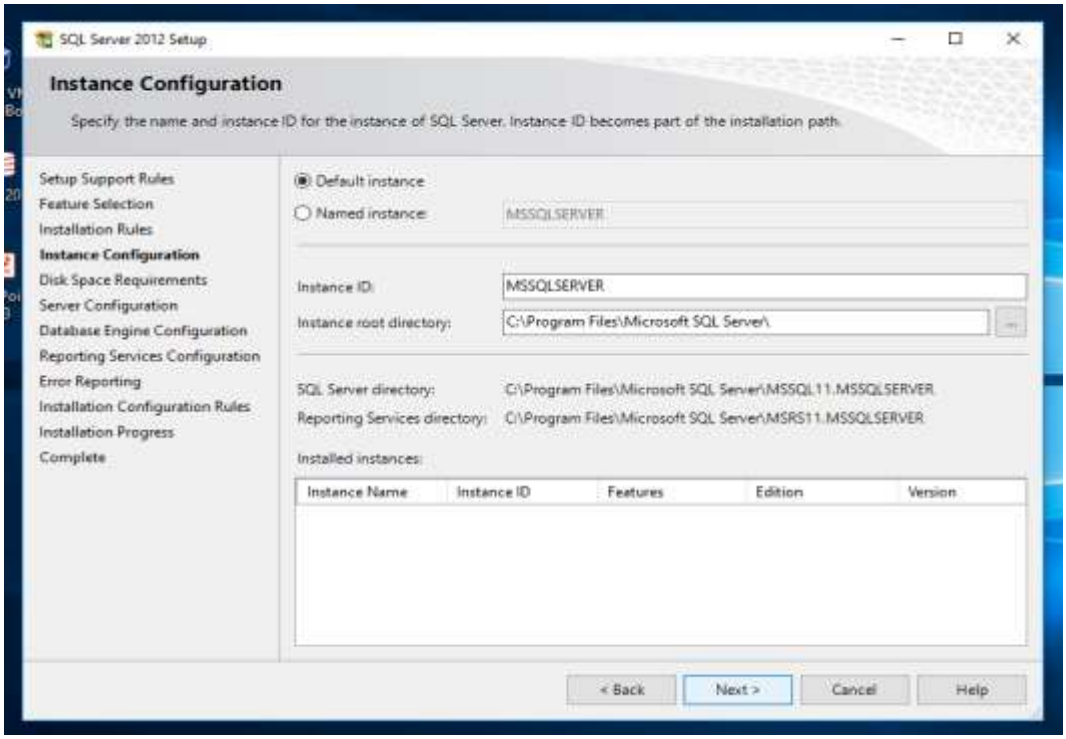
Gambar 41. *Feature Selection*

10) *Instance Configuration → pilih defauld instance*



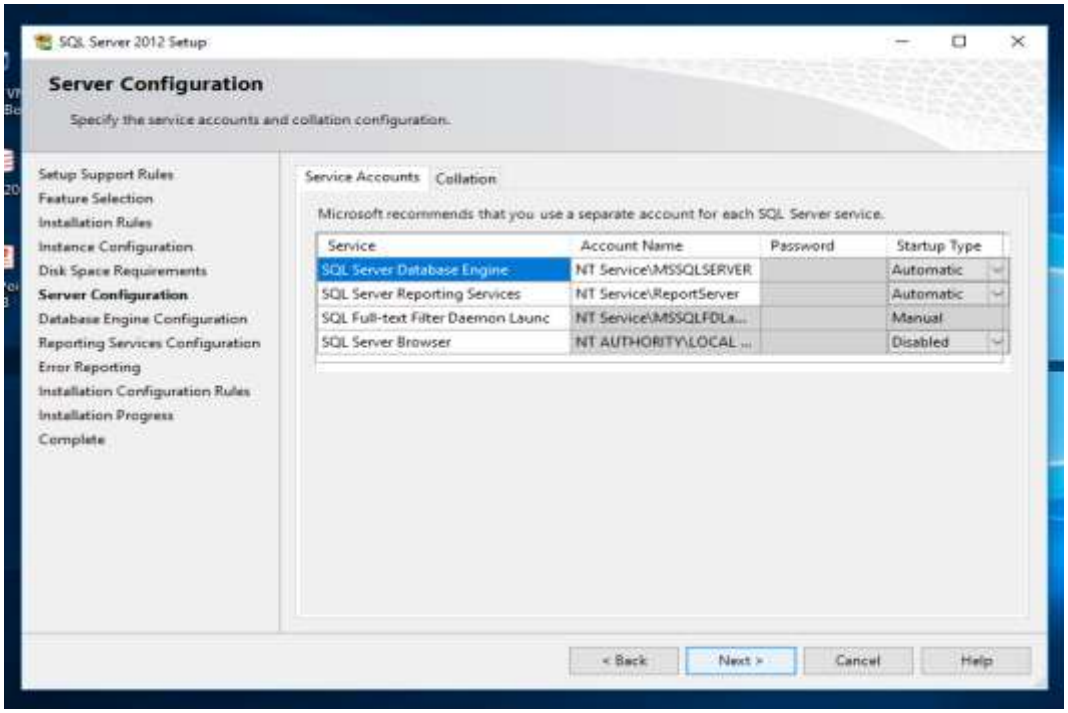
Gambar 42. *Instance Configuration*

11) *Instance Configuration → pilih defauld instance →klik Next*



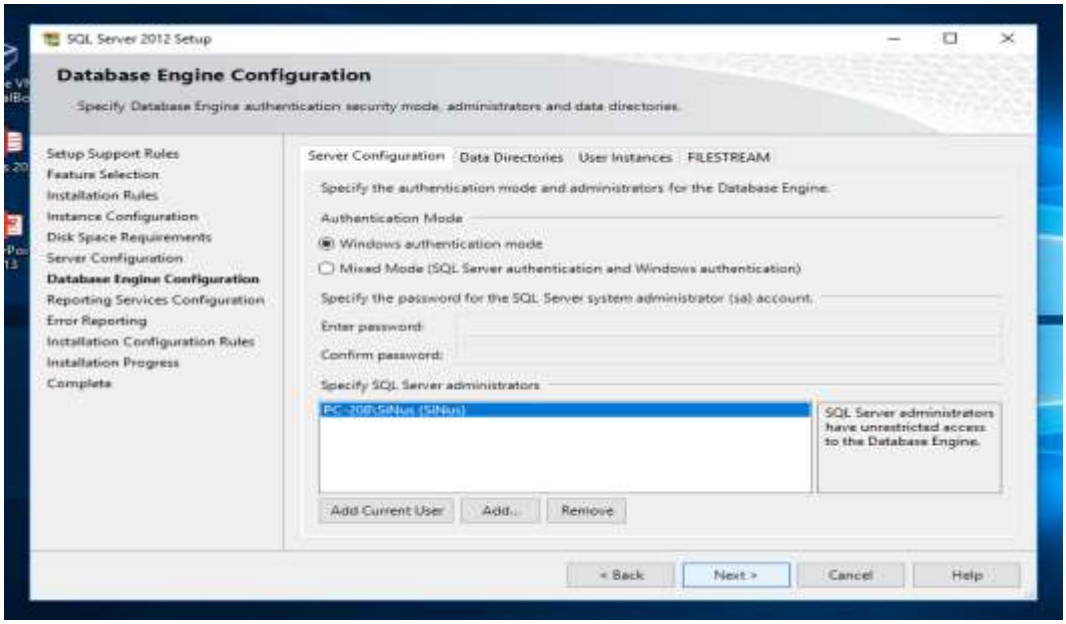
Gambar 43. *Instance Configuration*

12) Langkah selanjutnya adalah melakukan konfigurasi, *Server Configuration → SQL Server Database Enngine*



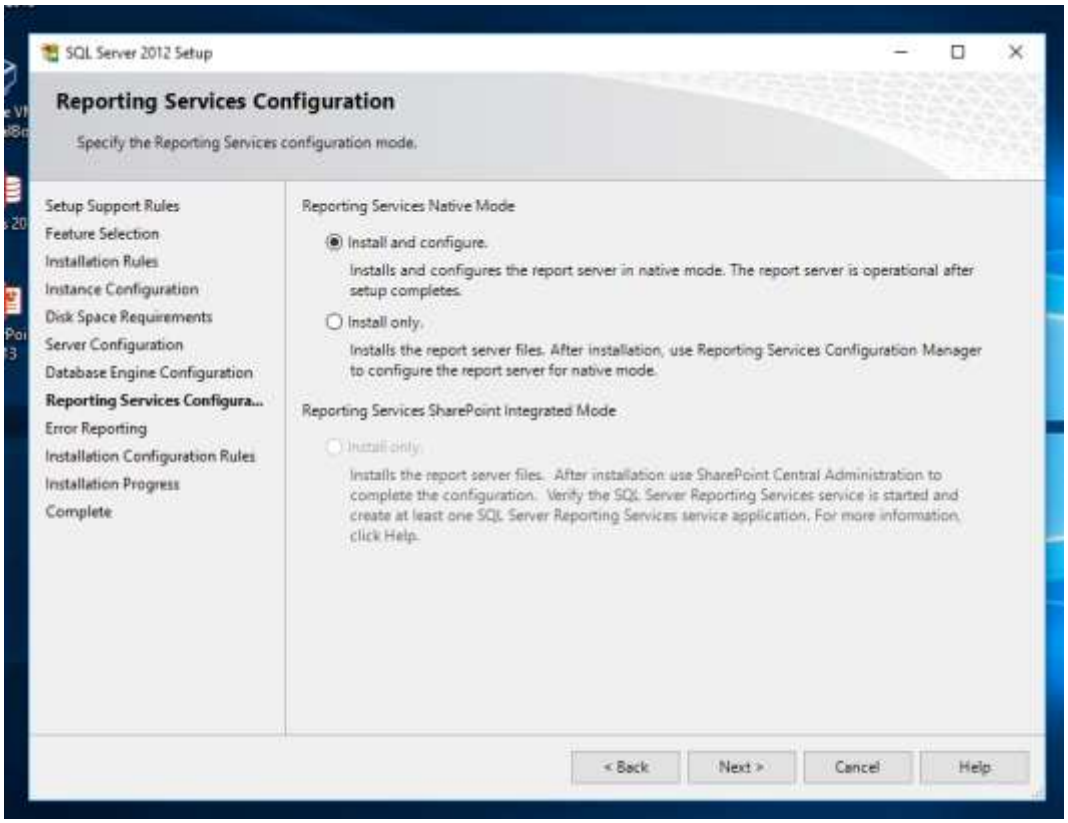
Gambar 44. *SQL Server Database Engine*

13) *SQL Server Database Engine Configuration → pilih Windows authentication mode → klik next*



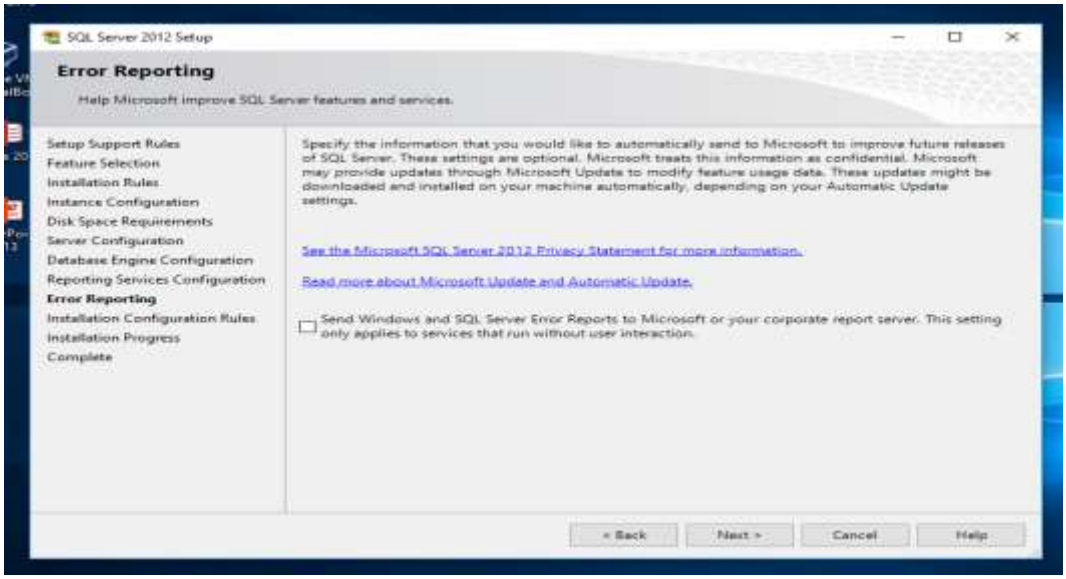
Gambar 45. *SQL Server Database Engine Configuration*

14) Langkah selanjutnya adalah *Reporting Services Configuration*



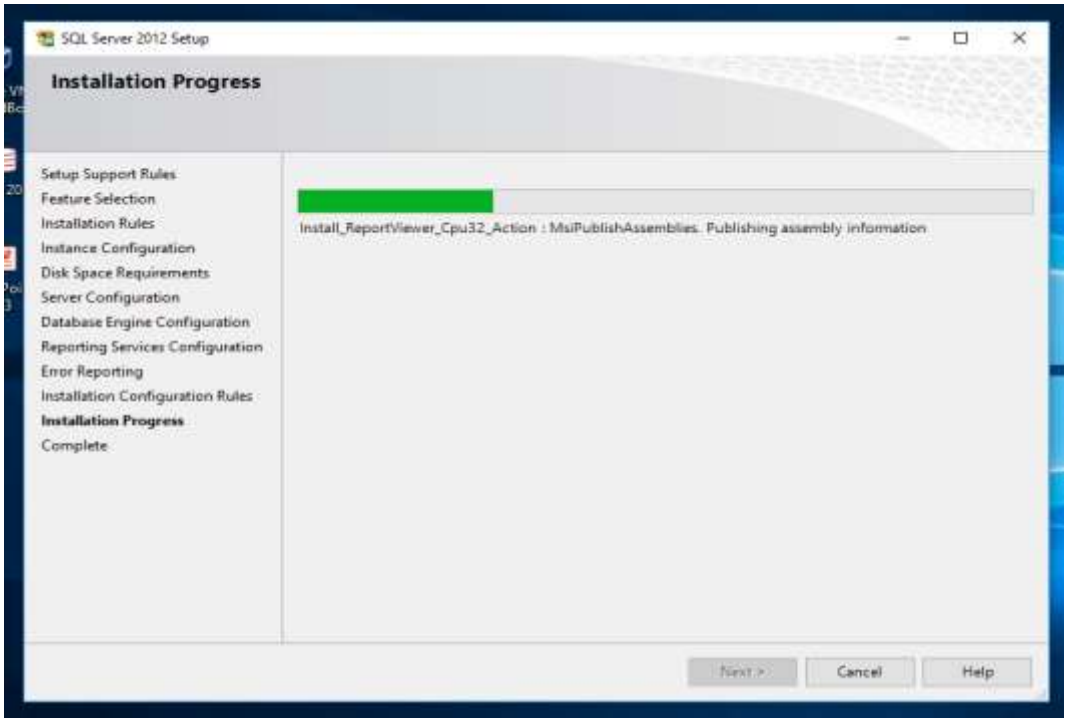
Gambar 46. *Reporting services Configuration*

15) Langkah selanjutnya adalah *Error Reporting* → *klik next*



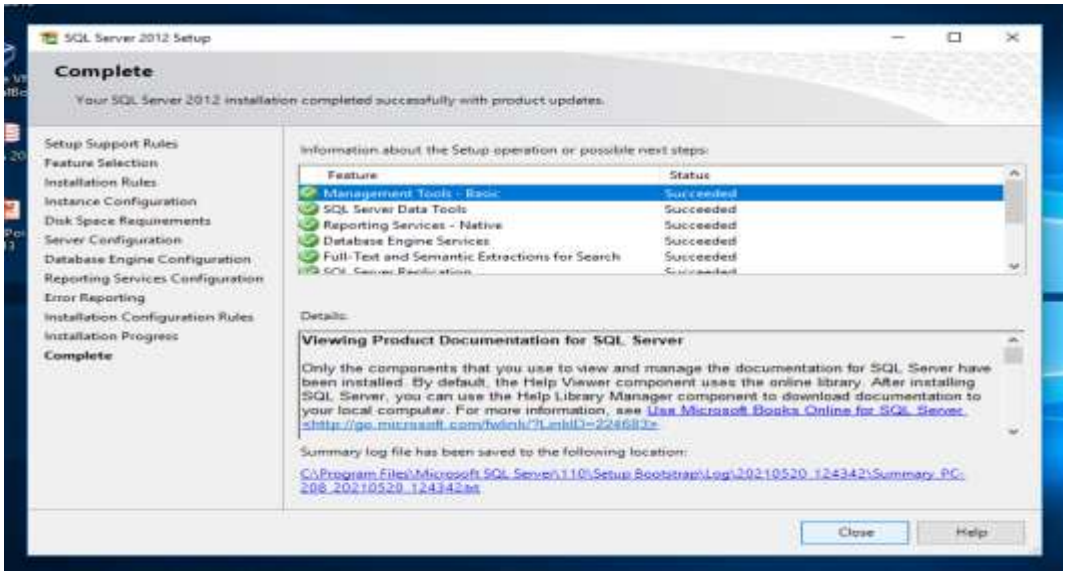
Gambar 47. *Error Reporting*

16) Langkah selanjutnya adalah *Installation Progress*, tunggu beberapa saat hingga proses selesai



Gambar 48. *Installation Progress*

17) SQL Server 2012 Setup *Complete* → *klik close*

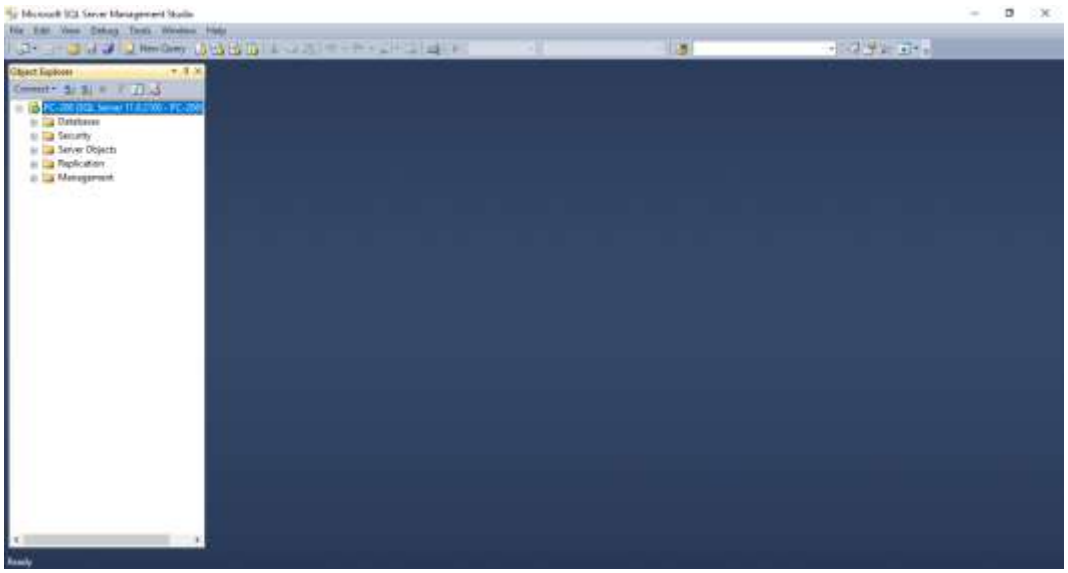


Gambar 49. *SQL Server 2012 setup Complete*

18) SQL Server Management Studio sudah terinstall dan siap dipergunakan → klik icon SQL Server Management Studii

Gambar 50. *SQL Server Management Studi siap dipergunakan*

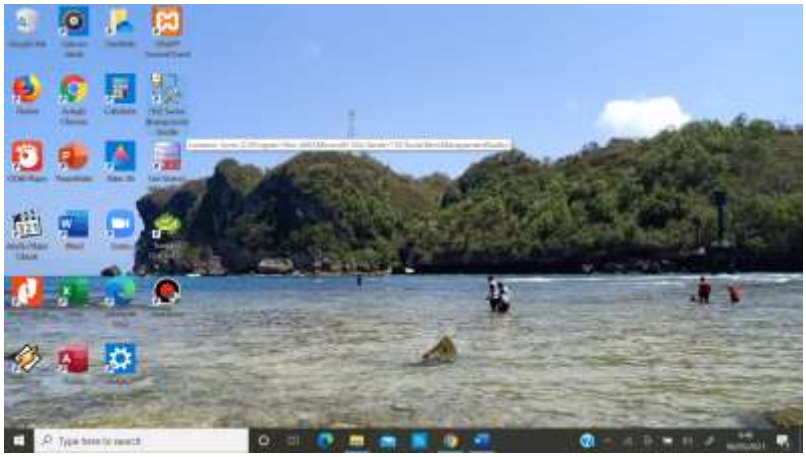
19) Sekarang Anda sudah dapat terhubung ke **SQL Server 2012 Ekspres** dan anda sudah dapat menggunakan SQL Server 2012 *Management Studio Express*.



Gambar 51. *SQL Server Management Studio sudah siap*

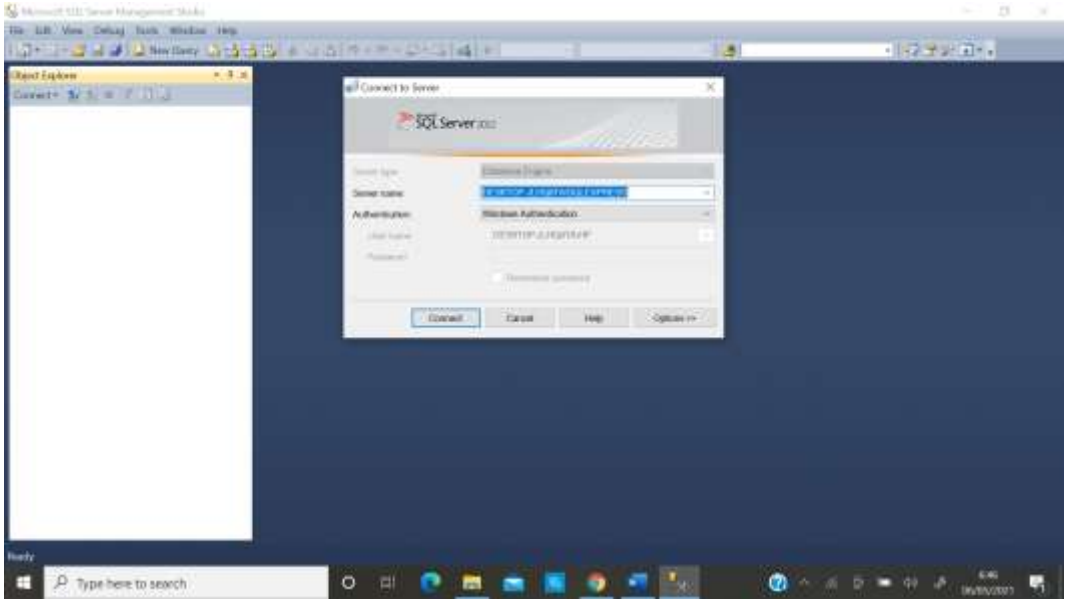
5.2 Menggunakan Sql Server 2012 Management Studio

1. Klik *SQl Server 2012 Management Studio*



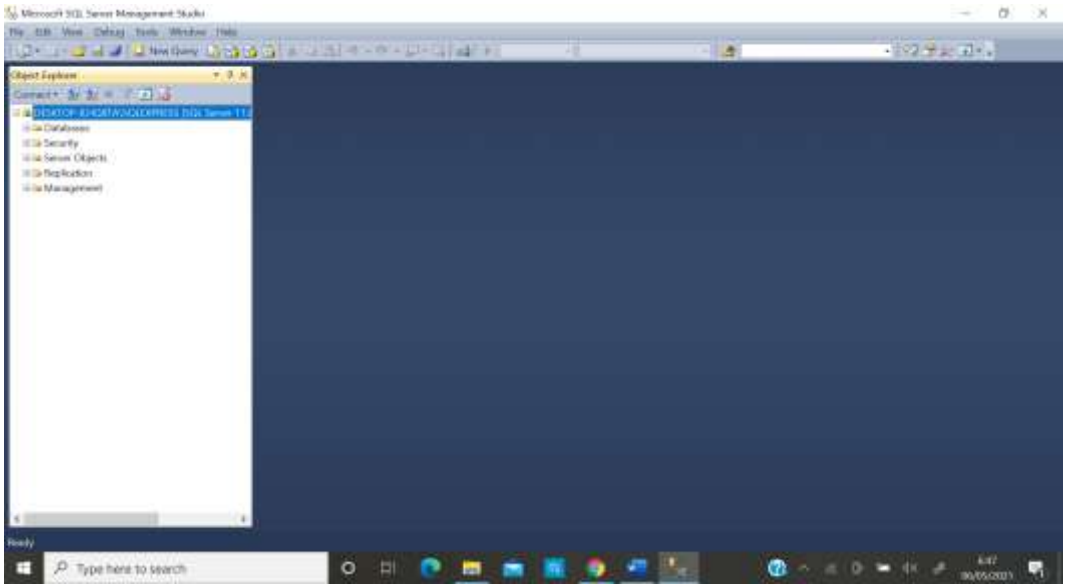
Gambar 52. *Klik Icon SQL Server di Desktop*

2. Kemudian akan muncul dialog layar seperti pada Gambar 53, agar tidak perlu memasukkan *user* dan pass, pilih *Authentication*: *Windows Authentication*, kemudian klik *Connect*.



Gambar 53. Tampilan Pilihan *Connect to Server*

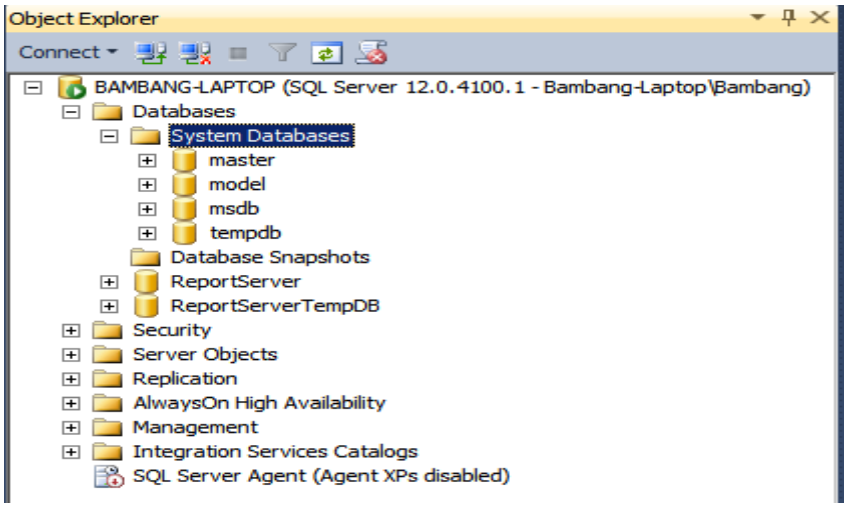
3. Setelah terkoneksi maka akan muncul tampilan layar seperti pada Gambar 54 *SQL Server 2012 Management Studio* siap untuk dipergunakan.



Gambar 54. *Tampilan SQL Server 2012 Management Studio*

5.3 Membuat Database Di SQL Server Managament Studio

Kita dapat membuat database di *SQL Server Managament Studio* dengan menggunakan *script* SQL atau menuliskannya secara langsung pada objek Databases. Sebagai tahap permulaan, kita akan membuat database secara langsung dengan menuliskannya di objek Database.



Gambar 55. Bagian *system database*

Sebelum memulai pembuatan database di SSMS, ada baiknya kita pahami terlebih dahulu apa itu System Database (Seperti pada Gambar 56). *System Database* adalah database yang digunakan oleh SQL Server untuk menangani hal-hal yang berkaitan dengan sistem pada SQL Server. Ada empat macam database yang merupakan bagian dari *System Database*. Berikut ini adalah penjelasannya:

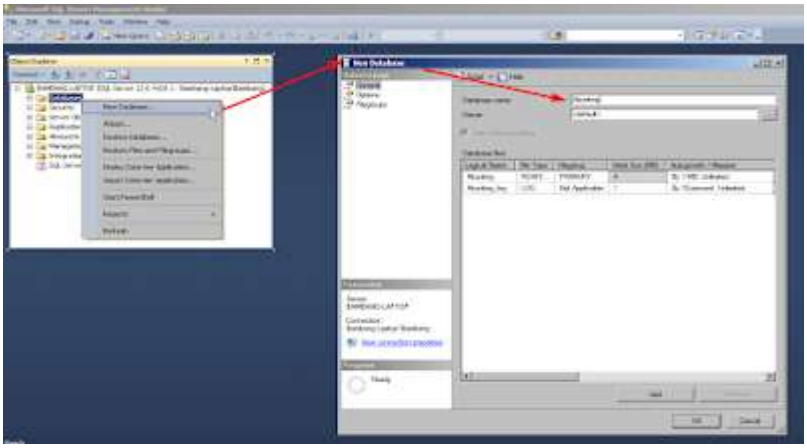
1. Database master: database ini menyimpan informasi level dari sistem yang ada pada SQL Server. Informasi level itu adalah akun pengguna, pengaturan konfigurasi, dan semua informasi yang berhubungan dengan database lainnya.
2. Database model: database ini digunakan sebagai template untuk semua database yang akan kita buat.
3. Database msdb: database ini digunakan oleh SQL Server Agent untuk mengatur semacam tugas yang beupa peringatan dan jadwal.
4. Database tempdb: database ini berisi semua tabel temporer, prosedur temporer dan semua penyimpanan yang sifatnya temporer. Database temporer ini diperlukan dan dihasilkan oleh SQL Server.

Masing-masing dari keempat database di atas mempunyai peran dan tugas s*end*iri-s*end*iri. Saat kita membuat database baru, misalnya. Database baru ini dibuat berdasarkan template yang disediakan oleh database model. Database model ini menyediakan template untuk semua database baru yang kita buat.

Membuat Database Baru

Berikut ini adalah langkah-langkah yang diperlukan dalam membuat database pada SQL Server 2014 menggunakan SSMS.

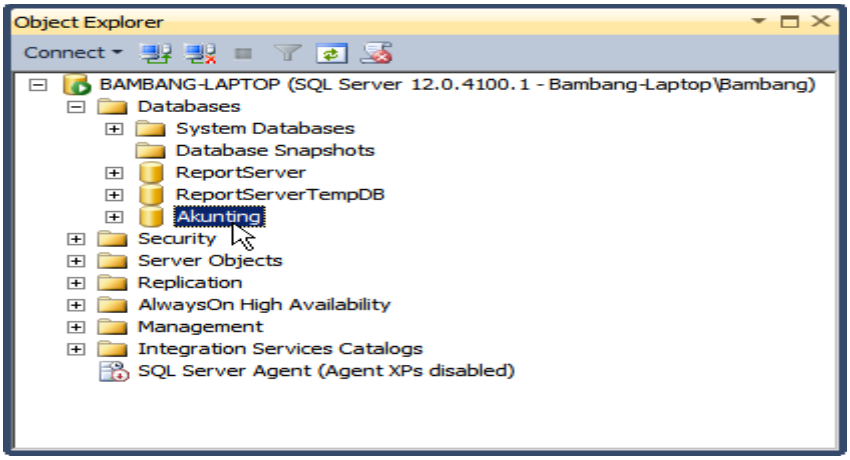
1. Dari *Object Explorer*, klik mouse kanan pada icon atau folder Database dan pilih New Database.



Gambar 56. Mambuat database baru

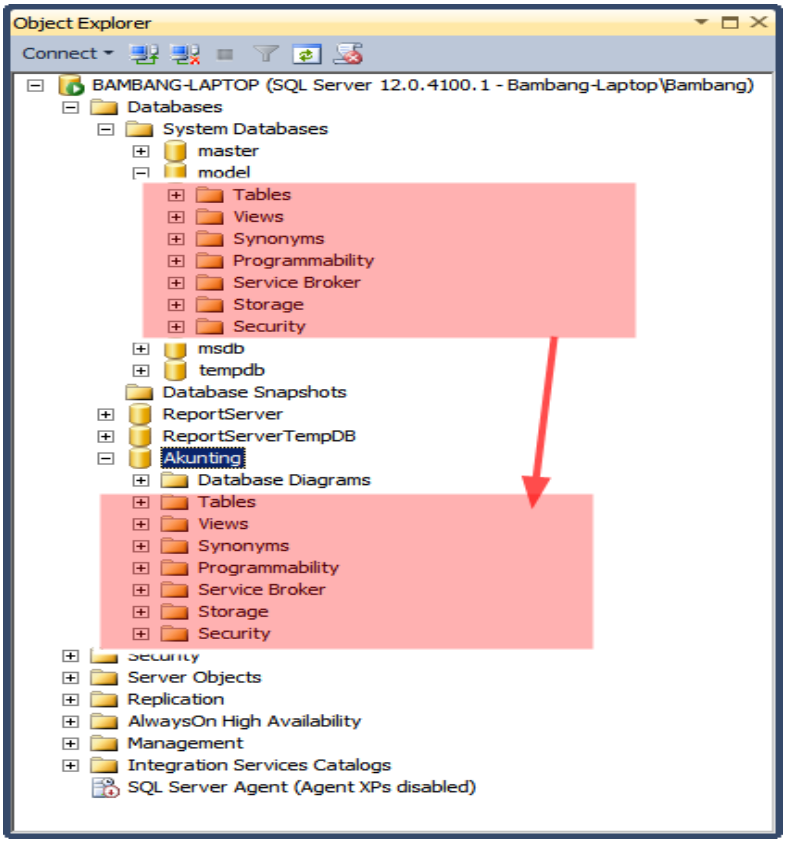
2. Beri nama database yang diinginkan
3. Klik OK bila sudah sesuai dengan keinginan.

Sangat mudah bukan? Ya, inilah cara paling mudah membuat database di SQL Server. Database baru yang kita buat akan ditempatkan di *object Database* dari *Object Explorer*.



Gambar 57. Database baru yang bernama Akunting

Seperti telah dijelaskan di atas, database baru dibuat dengan menggunakan databas model sebagai patokannya. Database model digunakan sebagai template saat sebuah database baru dibuat. Jika kita buka *folder System Database* lalu databas model, maka akan terlihat beberapa bagian dari objek database model yang digunakan sebagai template pada database baru, seperti terlihat pada Gambar 58.

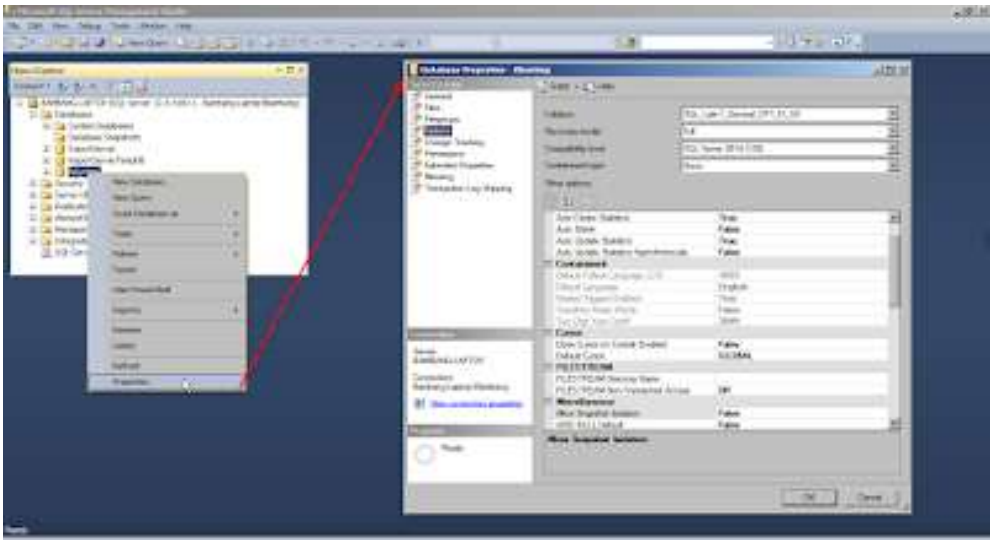


Gambar 58 Template database model pada database Akunting

Misalnya, pada database model ada tujuh folder yang terdiri dari Tables, Views, Synonyms, Programability, Service Broker, Storage, dan Security. Ketujuh folder itu akan selalu digunakan sebagai template untuk pembuatan database baru.

Kita baru saja membuat database baru dengan menggunakan pilihan default. Saat sebuah database baru dibuat, sebuah file data dan log transaksi juga dibuat. File data dan log transaksi ini dibuat di lokasi default dari server. Bila diinginkan, kita bisa mengubah lokasi file data dan log transaksi saat membuat database baru. Kita juga bisa mengubah berbagai macam pilihan spesifikasi dari database baru, seperti apakah ukuran database itu bisa bertambah secara otomatis sejalan dengan bertambahnya data. Bila demikian, kita bisa mengelola data yang bertambah itu.

Untuk mengubah spesifikasi database, kita dapat membuka properti dari database itu. Caranya, klik kanan mouse pada database dibuat, lalu pilih Properties pada menu shortcut, sehingga muncul kota dialog Properties. Ada banyak sekali parameter dari properti database yang bisa kita ubah sesuai dengan kebutuhan.



Gambar 59. Membuka properti database Akunting

Caranya, pilih saja pada salah satu tab di kotak Select a Page. Setiap tab pada kotak itu akan menampilkan properti yang terkait dengan tab itu. Bila semua properti database sudah dirasa benar, kita dapat menyimpannya dengan menekan tombol OK.

BAB 6. *QUERY*

Query adalah *syntax* atau perintah yang digunakan untuk mengakses dan menampilkan data pada sistem database. *Query* memiliki kemampuan untuk mengatur data mana yang perlu ditampilkan sesuai dengan yang Anda inginkan. Selain itu, query dapat dipakai untuk membuat data dapat saling berinteraksi. *Query* juga biasanya sering disebut dengan *query language* atau bahasa query. Saat ini bahasa *query* yang paling populer dikalangan *Database Administrator* adalah SQL. (Dewson, 2015)

Terdapat tiga jenis query database pada SQL, yaitu:

- DDL (*Data Definition Language*)
- DML (*Data Manipulation Language*)
- DCL (*Data Control Language*).

Ketiga query ini berfungsi untuk membuat/m*end*efinisikan objek-objek database seperti membuat tabel, memanipulasi database, dan mengontrol database.

6.1. Memahami Perintah DDL

DDL atau yang disebut dengan *Data Definition Language* yaitu kumpulan perintah pada SQL untuk menggambarkan desain dari database secara menyeluruh, selain itu DDL (*Data Definition Language*) juga digunakan untuk membuat, merubah maupun menghapus struktur atau definisi tipe data dari obyek yang ada pada database.

DDL adalah sebuah metode *query* SQL yang dipakai untuk mendefinisikan data di sebuah database. Dengan *query* inilah Anda dapat membuat tabel baru, mengubah tabel, membuat indeks, menentukan struktur penyimpanan tabel dan sebagainya. (Dewson, 2015). Tabel 9 adalah query yang dimiliki DDL.

Tabel 9. *Query* yang dimiliki DDL

Nama *Query*	Deskripsi
CREATE	Dipakai untuk membuat database dan tabel.
Drop	Dipakai untuk menghapus tabel dan database.
Alter	Dipakai untuk melakukan perubahan struktur tabel yang telah dibuat. Misalnya, menambah Field (***Add***), mengganti nama Field (***Change***) ataupun menamakannya kembali (***Rename***), dan menghapus Field (***Drop***).

Fungsi DDL (*Data Definition Language*)

DDL (Data Definition Language) merupakan perintah SQL yang berfungsi untuk membuat, merubah dan menghapus struktur data pada database. Perintah DDL antara lain:

- *Create database* berfungsi untuk membuat database baru.
- *Create Function* berfungsi untuk membuat fungsi pada database.
- *Create index* berfungsi untuk membuat index pada database.
- *Create procedure* berfungsi untuk membuat *procedure* pada database.
- *Create Table* yaitu perintah yang digunakan untuk membuat tabel baru pada database.
- *Create Trigger* berfungsi untuk membuat trigger pada database.

A. Perintah *CREATE*

a. Membuat Database

```
CREATE DATABASE nama_database
```

Contoh membuat database **TOKO**

```
CREATE DATABASE toko
```

b. Membuat Tabel

Database berisi table untuk menyimpan entitas. Table terdiri dari field (kolom) dan record (baris data). Perintah CREATE digunakan untuk membuat table.

```
CREATE TABLE nama_table
(
   field1 tipe_data,
   field2 tipe_data,
   field3 tipe_data
)
```

Contoh: Membuat Tabel Barang, yang terdiri atas field/kolom Kode Barang, Nama Barang, Jenis Barang, Jumlah barang, harga barang. Pilih New Query dan ketik perintah berikut ini:

```
CREATE TABLE tb_barang
(
   Kd_barang varchar (5),
   Nm_barang VARCHAR(50),
   Jenis_brg VARCHAR(50),
   Jumlah int,
   Harga numeric(18,0),
)
```

Kita telah membuat table **Barang**

```
INT(10) → tipe data integer dengan maksimal 10 digit angka
VARCHAR(100) → tipe data varchar dengan maksimal 100 karakter
```

Tipe data digunakan untuk mendefinisikan tipe dari *field* di *table*. Beberapa tipe data yang sering digunakan dalam SQLseperti pada Tabel 10.

Tabel 10. *Type data*

Tipe Data	Keterangan
INT	Menyimpan nilai integer

FLOAT	Menyimpan nilai float
VARCHAR	Menyimpan nil;ai string
CHAR	Menyimpan nilai satu karakter
DATE	Menyimpan nilai waktu
TEXT	Menyimpan nilai teks

PRIMARY KEY

PRIMARY KEY digunakan sebagai identifier unik untuk setiap *record* dan tidak boleh mengandung nilai *NULL*.

Cara penulisan:

```
CREATE TABLE nama_table
(
   field1 tipe_data,
   field2 tipe_data,
   field3 tipe_data,
   PRIMARY KEY(field)
)
```

Karena *PRIMARY KEY* harus bernilai unik maka untuk table mahasiswa yang bisa digunakan sebagai *PRIMARY KEY* adalah field NIM. Cara menuliskan perintah sebagai berikut:

```
CREATE TABLE tb_barang
(
   Kd_barang varchar (5) primary key,
   Nm_barang VARCHAR(50),
   Jenis_brg VARCHAR(50),
   Jumlah int,
   Harga numeric(18,0),
)
```

B. Perintah *ALTER*

Perintah ***ALTER*** berfungsi untuk merubah struktur tabel seperti menambah, merubah, menghapus kolom. *Alter Table* yaitu perintah yang digunakan untuk merubah struktur dari sebuah tabel.

Menambah kolom tabel

```
ALTER TABLE nama_table
ADD nama_field tipe_data
```

Contoh menambah *field* **Harga** di table **Barang**

```
ALTER TABLE tb_barang
ADD harga numeric (18,0)
```

Contoh menambah *field* Harga di table Barang

```
ALTER TABLE tb_barang
ADD jenis_brg varchar(50)
```

Modifikasi kolom tabele

```
ALTER TABLE nama_table
Alter column nama_field tipe_data
```

Contoh merubah tipe data **NM_BARANG** dari varchar(15) menjadi varchar(150)

```
ALTER TABLE TB_BARANG
ALTER COLUMN NM_BARANG VARCHAR(150)
```

Menghapus kolom tabel

```
ALTER TABLE nama_table
DROP column nama_field
```

Contoh menghapus *field* **jenis_barang**

```
alter table tb_barang
drop column jenis_brg
```

C. Perintah *TRUNCATE*

Perintah ***TRUNCATE*** digunakan untuk menghapus semua *record* di database.

```
TRUNCATE TABLE nama_table
```

Misal tabel **mahasiswa** telah mempunyai *record* dan kita ingin menghapus *record*nya.

```
TRUNCATE TABLE mahasiswa
```

D. Perintah *DROP*

Perintah ***DROP*** digunakan untuk menghapus tabel atau database

- *Drop Database* yaitu perintah yang berfungsi untuk menghapus database

  Jika ingin menghapus database, maka perintah yang digunakan adalah:

```
DROP DATABASE nama_database
```

Contoh: menghapus database toko, perintah yang diketikkan sebagai berikut:

```
DROP database toko
```

- *Drop Table* yaitu perintah yang digunakan untuk menghapus tabel pada database

  Jika ingin menghapus tabel, perintah yang digunakan:

```
DROP TABLE nama_table
```

  Contoh: Menghapus Tabel tb_barang, perintah yang diketikkan adalah:

```
DROP TABLE tb_barang
```

E. Perintah RENAME

Perintah *RENAME* digunakan untuk merubah nama tabel

```
RENAME TABLE nama_table_lama to nama_table_baru
```

Contoh merubah tabel **mahasiswa** menjadi **siswa**

```
RENAME TABLE mahasiswa to siswa
```

6.2. Memahami Perintah *Data Manipulation Language* (DML)

DML adalah sebuah metode *query* yang digunakan ketika DDL telah dibuat. *Query* DML ini dipakai untuk melakukan manipulasi database (Dewson, 2015). Tabel 11 adalah *query* yang dimiliki DML.

Tabel 11. *Query* yang dimiliki DML

Nama Query	Deskripsi
INSERT	Dipakai untuk memasukkan data pada tabel database.
UPDATE	Dipakai untuk mengubah data yang ada pada tabel database.
DELETE	Dipakai untuk menghapus data pada tabel database.

Data Manipulation Language (DML) adalah perintah SQL untuk manipulasi data dalam tabel. Perintah DML antara lain:

INSERT → menambah *record* di database

UPDATE → mengubah *record* di database

DELETE → menghapus *record* di database

Kalau DDL fokus ke operasi struktur tabel/ database, sedangkan DML lebih fokus kepada operasi *record* data.

A. Perintah *INSERT*

Perintah ***INSERT*** digunakan untuk menambah *record* data ke database.

Cara penulisan perintah *INSERT*

```
INSERT INTO nama_table (field1, field2, field3, ...)
VALUES (nilai1, nilai2, nilai3, ...);
```

Contoh :

- Penambahan data di tabel mahasiswa (yang dibuat di materi DDL)

```
INSERT INTO mahasiswa (nim, nama, alamat)
VALUES (20400010,"Setiyowati","Sukoharjo")
```

- Jika semua nilai *field* diisi maka kita bisa mengabaikan nama *field* di perintah *INSERT*.

```
INSERT INTO mahasiswa
VALUES (20400010,"Setiyowati","Sukoharjo")
```

- Jika hanya beberapa *field* saja yang ingin dimasukkan maka nama *field* juga harus ditulis secara spesifik. Misal hanya field **nim** dan **nama** saja yang akan di-*INSERT*.

```
INSERT INTO mahasiswa (nim, nama)
VALUES (20400010,"Setiyowati”)
```

- Jika ingin memasukkan beberapa *record* sekaligus dapat ditulis

```
INSERT INTO mahasiswa
VALUES
(20400010,"Setiyowati","Sukoharjo"),
(20400015,"Arif Zaini","Surakarta"),
(20400020,"Fatimah","Boyolali"),
(20400021,"Joko Susilo","Sragen");
```

B. Perintah *UPDATE*

Perintah ***UPDATE*** digunakan untuk merubah nilai *record* di database.

Cara penulisan perintah *UPDATE*

```
UPDATE nama_table
SET field1 = nilai1, field2 = nilai2, ...
WHERE kondisi;
```

Contoh : Jika kita ingin merubah alamat dari salah satu mahasiswa yang mempunyai NIM = 20400010

```
UPDATE mahasiswa
SET alamat = ‘Surakarta’
WHERE nim=’20400010’
```

Arti kode di atas adalah : kita melakukan update tabel **mahasiswa**. *Field* yang dirubah adalah **alamat** mahasiswa yang mempunyai nim “20400010” (pakai petik karena string) menjadi **Surakarta.**

C. Perintah DELETE

Perintah ***DELETE*** digunakan untuk menghapus *record* dari database

Cara penulisan perintah *DELETE*

```
DELETE
FROM nama_table
WHERE kondisi
```

Contoh : Menghapus *record* dengan NIM = 20400010

```
DELETE
FROM mahasiswa
WHERE nim='20400010'
```

6.3. Memahami Perintah *Data Query Language* (DQL)

DCL adalah metode *query* SQL yang dipakai untuk memberikan hak otorisasi akses database, auditan penggunaan database, alokasi *space*, dan definisi *space* (Dewson, 2015). Tabel 12 adalah *query* yang dimiliki DQL.

Tabel 12. *Query* yang dimiliki DQL

Nama *Query*	Deskripsi
GRANT	Dipakai untuk mengizinkan *user* mengakses tabel dalam database.
REVOKE	Dipakai untuk membatalkan izin hak *user*.
COMMIT	Dipakai untuk menetapkan penyimpanan database.

Data Query Language (DQL) adalah perintah SQL untuk query data.

Perintah DQL yaitu

```
SELECT → digunakan untuk melakukan query data dari database
```

Cara penulisan perintah SELECT

```
SELECT field1, field2, field3, ...

FROM nama_table;
```

Jika ingin membaca semua *field* tabel gunakan tanda *

```
SELECT *
FROM nama_table;
```

Misal kita telah membuat database **universitas** dengan tabel **Mahasiswa** seperti materi DDL dan mengisi nilai dengan materi DML.

```
INSERT INTO mahasiswa
VALUES
(20400010,'Setiyowati','Sukoharjo'),
(20400015,'Arif Zaini','Surakarta'),
(20400020,'Fatimah','Boyolali'),
(20400021,'Joko Susilo','Sragen');
```

Kita gunakan perintah *SELECT* untuk mengambil data mahasiswa dan menampilkannya di layer, hasilnya seperti pada Tabel 13.

```
SELECT * FROM mahasiswa;
```

Tabel 13. Hasil perintah *DQL* dari Tabel Mahasiswa

Nim	Nama	Alamat
20400010	Setiyowati	Sukoharjo
20400015	Arif Zaini	Surakarta
20400020	Fatimah	Boyolali
20400021	Joko Susilo	Sragen

Jika hanya ingin menampilkan beberapa atribut dari mahasiswa maka harus ditentukan field yang ingin di- *SELECT*, hasil seperti pada Tabel 14.

```
SELECT nama, alamat FROM mahasiswa;
```

Tabel 14. Hasil perintah *DQL* berdasarkan atribut dari Tabel Mahasiswa

Nama	Alamat
Setiyowati	Sukoharjo
Arif Zaini	Surakarta
Fatimah	Boyolali
Joko Susilo	Sragen

4 rows in set (0.00 sec)

Ada beberapa klausa dan operator penting untuk melakukan filter dan aggregasi dari perintah *SELECT* antara lain seperti pada Tabel 15.

Tabel 15. Klausal atau Operator perintah SELECT

Klausa / Operator	Makna
WHERE	Filter pencarian data
AND, OR, NOT	Operator pelengkap WHERE
LIKE	Operator pelengkap WHERE
ORDER BY	Mengurutkan data secara A-Z / Z-A
LIMIT	Membatasi hasil pencarian
AGGREGATION	Fungsi agregasi
GROUP BY	Mengelompokkan nilai berdasarkan field

a. *WHERE*

WHERE digunakan sebagai pelengkap *SELECT* untuk filter pencarian data.

```
SELECT field1, field2, field3, ...
FROM nama_table
WHERE kondisi
```

Misal mencari data mahasiswa yang bernama **Setiyowati**

```
SELECT * FROM mahasiswa WHERE nama='Setiyowati';
```

Hasil pencarian seperti pada Tabel 16.

Tabel 16. Hasil filter pencarian data

Nim	Nama	Alamat
20400010	Setiyowati	Sukoharjo

1 row in set (0.00 sec)

b. AND, OR, NOT

Klausa *WHERE* dapat dikombinasikan dengan operator *AND*, *OR* dan *NOT*.

Operator *AND* akan menghasilkan nilai *TRUE* jika kedua pernyataan bernilai benar.

```
SELECT field1, field2, field3, ...
FROM nama_table
WHERE kondisi1 AND kondisi2
```

Contoh, mencari data mahasiswa yang bernama **Setiyowati** dan beralamat **Sukoharjo.**

```
SELECT * FROM mahasiswa WHERE nama='Setiyowati' AND alamat= 'Sukoharjo';
```

Hasil perintah tersebut seperti pada Tabel 17.

Tabel 17. Hasil pencarian data dengan operator AND

Nim	Nama	Alamat
20400010	Setiyowati	Sukoharjo

1 row in set (0.00 sec)

Sekarang kita coba cari data mahasiswa yang bernama Setiyowati **dan** beralamat Jakarta.

```
SELECT * FROM mahasiswa WHERE nama='Setiyowati' AND alamat='Jakarta';
```

Empty set (0.00 sec)

Tidak m*end*apatkan nilai apapun karena saat pencarian terdapat 2 pernyataan yang bernilai *TRUE* dan *FALSE* **(mahasiswa Setiyowati di database hanya berasal dari Sukoharjo).**

Operator *OR* akan menghasilkan nilai *TRUE* jika salah satu pernyataan bernilai benar.

```
SELECT field1, field2, field3, ...
FROM nama_table
WHERE kondisi1 OR kondisi2
```

Jika kita mencari data mahasiswa yang bernama Setiyowati atau mahasiswa yang beralamat Boyolali.

```
SELECT * FROM mahasiswa WHERE nama='Setiyowati' OR alamat='Boyolali';
```

Hasil perintah tersebut seperti pada Tabel 18.

Tabel 18. Hasil pencarian data dengan operator OR

Nama	Alamat
Setiyowati	Sukoharjo
Fatimah	Boyolali

2 rows in set (0.00 sec)

Maka dengan operator OR akan dicari data mahasiswa yang bernama **Setiyowati** dan mahasiswa yang berasal dari **Boyolali**

Operator NOT akan menghasilkan nilai TRUE jika pernyataan bernilai salah.

```
SELECT field1, field2, field3, ...
FROM nama_table
WHERE NOT kondisi
```

Contoh : mencari data mahasiswa yang **tidak bernama Setiyowati**

```
SELECT * FROM mahasiswa WHERE NOT nama='Setiyowati';
```

Hasil perintah tersebut seperti pada Tabel 19.

Tabel 19. Hasil pencarian data dengan operator NOT

Nim	Nama	Alamat
20400015	Arif Zaini	Surakarta
20400020	Fatimah	Boyolali
20400021	Joko Susilo	Sragen

3 rows in set (0.00 sec)

c. LIKE

Klausa *WHERE* dapat dikombinasikan dengan operator *LIKE* untuk mencari data yang lebih spesifik berdasarkan pola

```
SELECT field1, field2, field3, ...
FROM nama_table
WHERE kondisi LIKE '%pola%';
```

Mencari data mahasiswa yang mempunyai nama dengan pola **'ar'**

```
SELECT * FROM mahasiswa WHERE nama LIKE '%ar%';
```

Hasil perintah tersebut seperti pada Tabel 20.

Tabel 20. Hasil pencarian data dengan operator *LIKE*

Nim	Nama	Alamat
20400015	Arif Zaini	Surakarta

1 rows in set (0.00 sec)

d. *ORDER BY*

ORDER BY digunakan sebagai pelengkap *SELECT* untuk mengurutkan data secara *ascending* (A-Z) atau *descending* (Z-A).

```
SELECT field1, field2, field3, ...
FROM nama_table
ORDER BY field1 | field2 | field3 ... ASC | DESC
```

Misal kita ingin menampilkan data mahasiswa berdasarkan nama terurut dari huruf A – Z

```
SELECT * FROM mahasiswa ORDER BY nama ASC;
```

Hasil perintah tersebut seperti pada Tabel 21.

Tabel 21. Hasil pencarian data dengan operator *ORDER BY*

Nim	Nama	Alamat
20400010	Setiyowati	Sukoharjo
20400015	Arif Zaini	Surakarta
20400020	Fatimah	Boyolali
20400021	Joko Susilo	Sragen

4 rows in set (0.00 sec)

Misal ingin menampilkan data mahasiswa yang mempunyai nama dengan pola ‘**ar**‘ dan terurut dari huruf Z – A dengan kolom **nama** dan **alamat** saja.

```
SELECT nama,alamat FROM mahasiswa WHERE nama LIKE '%ar%' ORDER BY nama DESC;
```

Hasil perintah tersebut seperti pada Tabel 22.

Tabel 22. Hasil pencarian data dengan operator *LIKE* dan *ORDER BY*

Nama	Alamat
Arif Zaini	Surakarta

1 rows in set (0.00 sec)

Menggabungkan klausa *WHERE* dan operator *LIKE* dengan *ORDER BY*.

Contoh lainnya untuk mengurutkan data berdasarkan 2 *field* / kolom. Misal data diurutkan berdasarkan field **nama** kemudian **alamat.**

```
SELECT * FROM mahasiswa ORDER BY nama,alamat ASC;
```

Hasil perintah tersebut seperti pada Tabel 23.

Tabel 23. Hasil pencarian data dengan operator *ORDER BY* berdasarkan 2 *field*

Nim	Nama	Alamat
20400015	Arif Zaini	Surakarta
20400020	Fatimah	Boyolali
20400021	Joko Susilo	Sragen
20400010	Setiyowati	Sukoharjo

4 rows in set (0.00 sec)

e. *LIMIT*

Keyword *LIMIT* digunakan untuk membatasi hasil pencarian

```
SELECT field1, field2, field3, ...
FROM nama_table
LIMIT N
```

Misal kita ingin mencari data mahasiswa yang terurut berdasarkan nama sebanyak 3 teratas.

```
SELECT * FROM mahasiswa ORDER BY nama ASC LIMIT 3;
```

Hasil perintah tersebut seperti pada Tabel 24.

Tabel 24. Hasil pencarian data dengan operator *LIMIT*

Nim	Nama	Alamat
20400015	Arif Zaini	Surakarta
20400020	Fatimah	Boyolali
20400021	Joko Susilo	Sragen

3 rows in set (0.00 sec)

f. AGGREGATION

Fungsi *AGGREGATION* digunakan untuk melakukan aggregasi data sehingga m*end*apatkan satu nilai berdasarkan hasil kalkulasi.

Ada beberapa fungsi yang banyak digunakan seperti ***COUNT(), MIN(), MAX(), AVG(), SUM().***

- *COUNT*() → menghitung jumlah data
- *MIN()* → mencari data terkecil
- *MAX()* → mencari data terbesar
- *AVG()* → mencari nilai rata-rata
- *SUM()* → mencari jumlah nilai

Sebelum melanjutkan ke *Hands On* kita akan tambah tabel mahasiswa (Tabel 25) dengan *field* UMUR dan setiap mahasiswa akan diberikan nilai UMUR (Tabel 26)

```
SELECT * FROM mahasiswa;
```

Tabel 25. Tabel mahasiswa

Nim	Nama	Alamat
20400010	Setiyowati	Sukoharjo
20400015	Arif Zaini	Surakarta
20400020	Fatimah	Boyolali
20400021	Joko Susilo	Sragen

4 rows in set (0.00 sec)

```
ALTER TABLE mahasiswa ADD umur INT(2);
Query OK, 0 rows affected (0.50 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

```
UPDATE mahasiswa SET umur=17 WHERE nim=20400010;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

```
UPDATE mahasiswa SET umur=18 WHERE nim=20400015;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

```
UPDATE mahasiswa SET umur=19 WHERE nim=20400020;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

```
UPDATE mahasiswa SET umur=20 WHERE nim=20400021;
Query OK, 1 row affected (0.06 sec)
Rows matched: 1 Changed: 1 Warnings: 0
```

```
SELECT * FROM mahasiswa;
```

Tabel 26. Tabel mahasiswa dengan tambahan *field* umur

Nim	**Nama**	**Alamat**	Umur
20400010	Setiyowati	Sukoharjo	17
20400015	Arif Zaini	Surakarta	18
20400020	Fatimah	Boyolali	19
20400021	Joko Susilo	Sragen	20

4 rows in set (0.01 sec)

Kita lanjut ke *query* data menggunakan fungsi agregasi. Cara menggunakan fungsi agregasi.

```
SELECT fungsi_agregasi(field)
FROM nama_table;
```

- Contoh *COUNT()*, mencari jumlah data mahasiswa

```
SELECT COUNT(*) FROM mahasiswa;
COUNT(*)
```

Hasilnya adalah

```
4
```

1 row in set (0.00 sec)

- Contoh MIN(), mencari umur terkecil mahasiswa

```
SELECT nama, MIN(umur) FROM mahasiswa;
```

Hasil seperti pada Tabel 27.

Tabel 27. Hasil pencarian dengan MIN()

Nama	Umur
Setiyowati	17

1 row in set (0.00 sec)

- Contoh MAX(), mencari umur terbesar mahasiswa

```
SELECT nama, MAX(umur) FROM mahasiswa;
```

Hasil seperti pada Tabl 28.

Tabel 28. Hasil pencarian dengan MAX()

Nama	Umur
Joko Susilo	20

1 row in set (0.00 sec)

- Contoh AVG(), mencari rata-rata umur mahasiswa

```
SELECT AVG(umur) FROM mahasiswa;
AVG(umur)
18.5000
```

1 row in set (0.00 sec)

- Contoh SUM(), mencari jumlah umur mahasiswa

```
SELECT sum(umur) FROM mahasiswa;
sum(umur)
```

Hasilnya adalah

```
74
```

1 row in set (0.00 sec)

g. *GROUP BY*

GROUP BY digunakan untuk mengelompokkan nilai yang sama berdasarkan field *GROUP BY* biasanya digunakan dengan fungsi agregasi

Cara penulisan *GROUP BY*

```
SELECT fungsi_agregasi(field)
FROM nama_table
GROUP BY field;
```

Sebelum lanjut ke *Hands On* kita akan tambah data mahasiswa terlebih dahulu agar dapat digunakan perintah *GROUP BY*'

```
INSERT INTO mahasiswa
VALUES
(21400205,"Rudiyanto","Bandung",17),
(21400206,"Setyawan","Bandung",17),
(21400207,"Siswanto","Semarang",19),
(21400208,"Budiman","Semarang",19),
(21400209,"Iqbal","Kalimantan",19),
(21400210,"Hasan","Malang",18),
(21400211,"Wahyudi","Malang",18);
Query OK, 7 rows affected (0.04 sec)
Records: 7 Duplicates: 0 Warnings: 0
```

```
SELECT * FROM mahasiswa;
```

Hasil perintah tersebut seperti pada Tabel 29.

Tabel 29. Hasil penambahan data

Nim	**Nama**	**Alamat**	Umur
20400010	Setiyowati	Sukoharjo	17
20400015	Arif Zaini	Surakarta	18
20400020	Fatimah	Boyolali	19
20400021	Joko Susilo	Sragen	20
21400205	Rudiyanto	Bandung	17

Nim	Nama	Alamat	Umur
21400206	Setyawan	Bandung	17
21400207	Siswanto	Semarang	19
21400208	Budiman	Semarang	19
21400209	Iqbal	Kalimantan	19
21400210	Hasan	Malang	18
21400211	Wahyudi	Malang	18

11 rows in set (0.00 sec)

- Contoh 1, menghitung jumlah mahasiswa berdasarkan alamat

```
SELECT alamat, COUNT(*) FROM mahasiswa GROUP BY alamat;
```

Hasil perintah tersebut seperti pada Tabel 30.

Tabel 30. Hasil perintah menghitung jumlah mahasiswa dengan *GROUP BY*

Alamat	COUNT(*)
Sukoharjo	1
Surakarta	1
Boyolali	1
Sragen	1
Bandung	2
Semarang	2
Kalimantan	1
Malang	2

8 rows in set (0.00 sec)

- Contoh 2, Mengurutkan data hasil *GROUP BY* berdasarkan alamat yang paling banyak.

```
SELECT alamat, COUNT(*) as jumlah FROM mahasiswa GROUP BY
alamat ORDER BY jumlah DESC;
```

Hasil perintah tersebut seperti pada Tabel 31.

Tabel 31. Hasil Mengurutkan data hasil *GROUP BY*

Alamat	Jumlah
Bandung	2
Malang	2

Semarang	2
Boyolali	1
Kalimantan	1
Sukoharjo	1
Surakarta	1
Sragen	1

8 rows in set (0.00 sec)

- Contoh 3, Mencari rata-rata umur mahasiswa berdasarkan alamat mahasiswa dan diurutkan berdasarkan nilai paling banyak.

```
SELECT alamat, AVG(umur) as rata_umur FROM mahasiswa GROUP BY alamat
ORDER BY rata_umur DESC;
```

Hasil perintah tersebut seperti pada Tabel 32.

Tabel 32. Hasil mencari rata-rata umur mahasiswa berdasarkan alamat

Alamat	Rata Umur
Sukoharjo	17
Surakarta	18
Boyolali	19
Sragen	20
Bandung	17
Semarang	19
Kalimantan	19
Malang	18

8 rows in set (0.00 sec)

6.4. Memahami Perintah *Data Control Language* (DCL)

Data Control Language (DCL) adalah perintah SQL untuk control dan *permission* database. Perintah DCL antara lain:

```
GRANT → Memberikan hak akses / hak istemewa pengguna
REVOKE → Menarik hak akses pengguna yang diberikan lewat perintah GRANT
```

A. Perintah *GRANT*

Perintah ***GRANT*** memungkinkan pemberikan hak akses kepada pengguna tidak harus setiap pengguna database dapat mengakses seluruh data di database. Ada pengguna yang hanya dapat melakukan operasi di satu table saja. Bisa juga pengguna hanya dapat melakukan operasi SELECT saja tanpa bisa melakukan manipulasi data, sehingga **diperlukan manajemen hak akses dengan *GRANT*** Sebelum kita menggunakan perintah GRANT terlebih dulu kita akan buat *user* di database dengan perintah

```
CREATE USER 'nama_user'@'localhost' IDENTIFIED BY 'password';
```

Note: untuk membuat *user* baru anda harus masuk sebagai **root**. Pahami cara masuk MySQL/MariaDB melalui terminal / command prompt menggunakan *user* **root** di Tutorial MySQL untuk pemula.

Misal kita akan bikin *user* **sinus** dengan password **12345**

CREATE *USER* 'sinus'@'localhost' IDENTIFIED BY '12345';

Query OK, 0 rows affected (0.00 sec)

Login ke MySQL dengan *user*name **sinus**

```
$ ./mysql -u sinus -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 209
Server version: 10.1.38-MariaDB Source distribution
Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.
Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.
MariaDB [(none)]>
```

Saat akan melihat list database, *user* **sinus** tidak dapat melihat keseluruhan data (berbeda jika melihat database dengan akses **root**)

show databases;

```
----------------------------------
| Database                       |
```

```
----------------------------------
| information_schema             |
| test                           |
| test_db_login                  |
| test_ecommerce_codeigniter |
```

Saat akan membuat database akan ditolak juga

```
CREATE DATABASE universitas;
```

ERROR 1044 (42000): Access denied for *user* 'sinus'@'localhost' to database 'universitas'

Hal ini dikarenakan *user* **sinus** tidak punya *privilege* terhadap database.
Mari kita beri akses dengan perintah *GRANT* melalui akses ***root*** dengan perintah

```
GRANT ALL PRIVILEGES ON * . * TO 'sinus'@'localhost';
```

Tanda "ALL PRIVILEGES" dan asterik (*) artinya *user* **sinus** diberi akses untuk melakukan semua operasi seperti menambah, mengubah atau menghapus data di semua table / database

Coba masuk kembali dengan *user* **sinus** dan buatlah database / table baru maka tidak akan bermasalah lagi. Beberapa opsi perintah GRANT seperti pada Tabel 33.

Tabel 33. Opsi perintah GRANT

TIPE IZIN	KETERANGAN
ALL PRIVILEGES	Memberikan akses full
CREATE	Memberikan akses membuat table / database
DROP	Memberikan akses menghapus table / database
SELECT	Memberikan akses menambah *record* di table
INSERT	Memberikan akses merubah *record* di table
UPDATE	Memberikan akses menghapus *record* di table
DELETE	Memberikan akses menggunakan perintah SELECT

Cara menggunakannya adalah

```
GRANT tipe_izin ON nama_database.nama_table TO 'nama_user'@'localhost';
```

Contoh *GRANT SELECT*

Masuk ke *user* ***root*** dan berikan hak akses *SELECT* saja untuk *user* **sinus**

```
GRANT SELECT ON *.* TO 'sinus'@'localhost';
```

Masuk ke *user* **sinus** dan coba lakukan operasi ***SELECT*** dan ***INSERT***

```
SELECT * FROM mahasiswa;
```

Nim	Nama	Alamat
20400010	Setiyowati	Sukoharjo
20400015	Arif Zaini	Surakarta
20400020	Fatimah	Boyolali
20400021	Joko Susilo	Sragen

4 rows in set (0.00 sec)

```
INSERT INTO mahasiswa values (222,"Jaka","Yogyakarta");

ERROR 1142 (42000): INSERT command denied to user 'sinus'@'localhost' for table 'mahasiswa'
```

Operasi *SELECT* dibolehkan sedangkan operasi *INSERT* tidak bisa dilakukan

Contoh *GRANT SELECT*, *INSERT, UPDATE, DELETE*

Jika misalkan *user* hanya dibolehkan untuk melakukan perintah DML dan DQL tanpa akses merubah struktur table / database maka

```
GRANT SELECT,INSERT, UPDATE, DELETE ON *.* TO 'sinus'@'localhost';
```

B. Perintah *REVOKE*

Perintah *REVOKE* digunakan untuk mencabut kembali hak akses yang diberikan melalui perintah *GRANT*

Cara menggunakannya

```
REVOKE tipe_izin ON nama_database.nama_table FROM 'username'@'localhost';
```

Jika kita ingin menghapus akses INSERT di *user* **sinus** untuk semua database dan table melalui **root**

```
REVOKE INSERT ON *.* FROM 'ngodingdata'@'localhost';
```

Masuk menggunkan *user* **sinus** dan coba masukkan *record* baru

```
use universitas;
Database changed
INSERT INTO mahasiswa values (223,"Dika","Semarang");
ERROR 1142 (42000): INSERT command denied to user sinus'@'localhost' for table
'mahasiswa'
```

Untuk mencabut seluruh hak akses *user* dapat menggunakan *REVOKE ALL*

```
REVOKE ALL ON nama_database.nama_table FROM 'username'@'localhost';
```

6.5. Memahami Perintah TCL

Transaction Control Language (TCL) adalah perintah SQL yang berhubungan dengan transaksi di database. Perintah TCL antara lain:

```
COMMIT → Menyimpan transaksi secara permanen

ROLLBACK →Mengembalikan database ke bentuk awal / COMMIT terakhir
```

A. Perintah COMMIT

Perintah *COMMIT* digunakan untuk menyimpan transaksi secara permanen di database. Saat melakukan perintah DML seperti *INSERT*, *UPDATE*, *DELETE* traksaksi sebenarnya belum dilakukan secara permanen. Artinya operasi tersebut masih bisa di rollback / di batalkan. Jika ingin menyimpan transaksi sehingga tidak dapat di rollback kita gunakan perintah *COMMIT*.

Kapan perintah *COMMIT* dibutuhkan?

Dalam suatu rangkaian operasi data, jika ada 1 atau lebih operasi yang mengalami kegagalan maka kita akan mengembalikan seperti ke bentuk semula. **Jika tidak ada kesalahan maka seluruh rangkaian pernyataan akan di – *COMMIT* untuk menyimpan transaksi secara permanen.**

Gunakan Database **universitas** dan Tabel **mahasiswa** yang telah kita buat di materi DDL.

```
INSERT INTO mahasiswa
VALUES
(20400010,"Setiyowati","Sukoharjo"),
(20400015,"Arif Zaini","Surakarta"),
(20400020,"Fatimah","Boyolali"),
```

```
(20400021,"Joko Susilo","Sragen");
```

Selanjutnya kita cek data yang telah diinputkan

```
SELECT * FROM mahasiswa;
```

Nim	Nama	Alamat
20400010	Setiyowati	Sukoharjo
20400015	Arif Zaini	Surakarta
20400020	Fatimah	Boyolali
20400021	Joko Susilo	Sragen

4 rows in set (0.00 sec)

Untuk memulai menggunakan COMMIT harus dimulai dengan

```
START TRANSACTION;

Query OK, 0 rows affected (0.00 sec)

INSERT INTO mahasiswa VALUES (21400210,"Jaka","Kalimantan");
Query OK, 1 row affected (0.00 sec)
COMMIT;
Query OK, 0 rows affected (0.07 sec)
```

```
SELECT * FROM mahasiswa;
```

Hasil perintah tersebut seperti Tabel 34.

Tabel 34. Hasil perintah COMMIT

Nim	Nama	Alamat
20400010	Setiyowati	Sukoharjo
20400015	Arif Zaini	Surakarta
20400020	Fatimah	Boyolali
20400021	Joko Susilo	Sragen
21400210	Jaka	Kalimantan

5 rows in set (0.00 sec)

B. Perintah *ROLLBACK*

Perintah *ROLLBACK* digunakan untuk mengembalikan database ke bentuk awal / *COMMIT* terakhir.

Perintah *COMMIT* dan *ROLLBACK* saling berkaitan

Kapan perintah *ROLLBACK* dibutuhkan?

Dalam suatu rangkaian operasi data, **jika ada 1 atau lebih operasi yang mengalami kegagalan maka kita akan mengembalikan seperti ke bentuk semula** menggunakan perintah *ROLLBACK.*

Untuk menggunakan *COMMIT* / *ROLLBACK* harus dimulai dengan *START TRANSACTION*;

```
START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
SELECT * FROM mahasiswa;
```

Nim	Nama	Alamat
20400010	Setiyowati	Sukoharjo
20400015	Arif Zaini	Surakarta
20400020	Fatimah	Boyolali
20400021	Joko Susilo	Sragen
21400210	Jaka	Kalimantan

5 rows in set (0.00 sec)

```
INSERT INTO mahasiswa VALUES (21400211,"Fitri","Surabaya");
```

Query OK, 1 row affected (0.00 sec)

```
SELECT * FROM mahasiswa;
```

Nim	Nama	Alamat
20400010	Setiyowati	Sukoharjo
20400015	Arif Zaini	Surakarta
20400020	Fatimah	Boyolali
20400021	Joko Susilo	Sragen

21400210	Jaka	Kalimantan
21400211	Fitri	Surabaya

6 rows in set (0.00 sec)

```
ROLLBACK;
```

Query OK, 0 rows affected (0.11 sec)

```
SELECT * FROM mahasiswa;
```

Nim	Nama	Alamat
20400010	Setiyowati	Sukoharjo
20400015	Arif Zaini	Surakarta
20400020	Fatimah	Boyolali
20400021	Joko Susilo	Sragen
21400210	Jaka	Kalimantan

5 rows in set (0.00 sec)

Saat kita gunakan perintah *ROLLBACK* akan kembali ke database awal.

6.6. Cara Menggunakan *JOIN*

JOIN merupakan perintah di MySQL untuk menggabungkan 2 table atau lebih berdasarkan kolom yang sama. *JOIN* di MySQL dibagi menjadi 3 cara:

1. INNER JOIN
2. LEFT JOIN
3. RIGHT JOIN

Sebelum kita melanjutkan pembahasan tentang join kita akan membuat 2 tabel yaitu table **mahasiswa** dan **transaksi**

Table mahasiswa → Data mahasiswa

Table transaksi → Data transaksi peminjaman buku perpus

```
CREATE TABLE mahasiswa
(
   nim INT(10),
```

```
   nama VARCHAR(100),
   alamat VARCHAR(100),
   PRIMARY KEY(nim)
);

CREATE TABLE transaksi
(
   id_transaksi INT(10),
   nim INT(10),
   buku VARCHAR(100),
   PRIMARY KEY(id_transaksi)
);
```

Masukkan data untuk tabel **mahasiswa** dan **transaksi**

```
INSERT INTO mahasiswa
VALUES
(21400200,"faqih","bandung"),
(21400201,"ina","jakarta"),
(21400202,"anto","semarang"),
(21400203,"dani","padang"),
(21400204,"jaka","bandung"),
(21400205,"nara","bandung"),
(21400206,"senta","semarang");
```

```
INSERT INTO transaksi
VALUES
(1,21400200,"Buku Informatika"),
(2,21400202,"Buku Teknik Elektro"),
(3,21400203,"Buku Matematika"),
(4,21400206,"Buku Fisika"),
(5,21400207,"Buku Bahasa Indonesia"),
(6,21400210,"Buku Bahasa Daerah"),
(7,21400211,"Buku Kimia");
SELECT * FROM mahasiswa;
```

Nim	Nama	Alamat
21400200	faqih	bandung
21400201	ina	jakarta
21400202	anto	semarang

21400203	dani	padang
21400204	jaka	bandung
21400205	nara	bandung
21400206	senta	semarang

7 rows in set (0.00 sec)

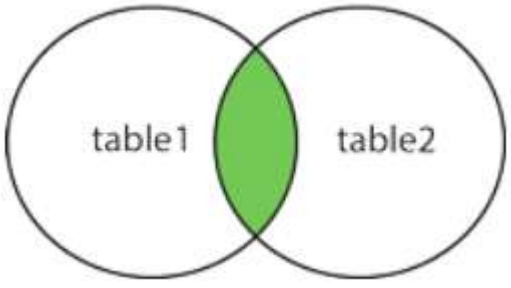
```
SELECT * FROM transaksi;
```

id_transaksi	nim	buku
1	21400200	Buku Informatika
2	21400202	Buku Teknik Elektro
3	21400203	Buku Matematika
4	21400206	Buku Fisika
5	21400207	Buku Bahasa Indonesia
6	21400210	Buku Bahasa Daerah
7	21400211	Buku Kimia

7 rows in set (0.00 sec)

A. *INNER JOIN*

INNER JOIN membandingkan *record* di setiap tabel untuk dicek apakah nilai sama atau tidak.



Gambar 60. *INNER JOIN*

Jika nilai kedua tabel sama maka akan terbentuk tabel baru yang hanya menampilkan *record* yang sama dari kedua tabel.

Cara penulisannya

```
SELECT *
FROM table1
INNER JOIN table2
ON table1.field = table2.field;
```

Contoh, Mencari data dari tabel mahasiswa dan tranksaksi berdasarkan kolom NIM

```
SELECT *
FROM mahasiswa
INNER JOIN transaksi
ON mahasiswa.nim = transaksi.nim;
```

Hasil perintah tersebut seperti pada Tabel 35.

Tabel 35. Hasil pencarian data dengan INNER JOIN

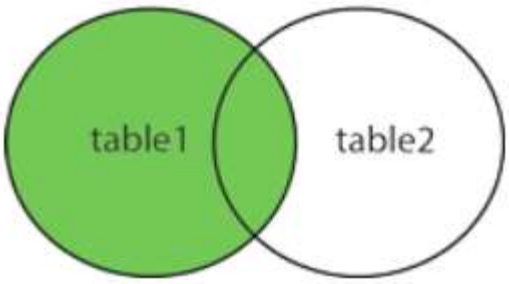
nim	nama	alamat	id_transaksi	nim	buku
21400200	faqih	bandung	1	21400200	Buku Informatika
21400202	anto	semarang	2	21400202	Buku Teknik Elektro
21400203	dani	padang	3	21400203	Buku Matematika
21400206	senta	semarang	4	21400206	Buku Fisika

4 rows in set (0.00 sec)

Tabel mahasiswa mempunyai 7 *record* dan table transaksi mempunyai 7 *record* Jika menggabungkan kedua data menggunakan INNER JOIN berdasarkan kolom NIM maka hanya tampil 4 data mahasiswa yang meminjam buku di perpustakaan.

B. LEFT JOIN

LEFT JOIN menghasilkan nilai berdasarkan tabel kiri (table1) dan nilai yang sama di tabel kanan (table2).



Gambar 61. LEFT JOIN

Jika tabel kanan tidak nilainya ada maka akan diisi nilai *NULL*.

```
SELECT *
FROM table1
LEFT JOIN table2
ON table1.field = table2.field;
```

Contoh, Mencari data dari table mahasiswa dan tranksaksi berdasarkan kolom

```
NIM
SELECT *
FROM mahasiswa
LEFT JOIN transaksi
ON mahasiswa.nim = transaksi.nim;
```

Hasil perintah tersebut seperti pada Tabel 36.

Tabel 36. pencarian data dengan LEFT JOIN

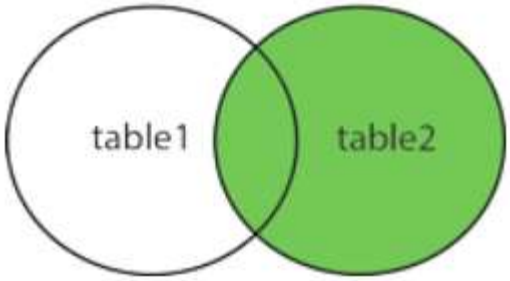
nim	nama	alamat	id_transaksi	nim	buku
21400200	faqih	bandung	1	21400200	Buku Informatika
21400202	anto	semarang	2	21400202	Buku Teknik Elektro
21400203	dani	padang	3	21400203	Buku Matematika
21400206	senta	semarang	4	21400206	Buku Fisika
21400201	Ina	Jakarta	Null	Null	Null
21400204	Jaka	Bandung	Null	Null	Null
21400205	Nara	Bandung	Null	Null	Null

7 rows in set (0.00 sec)

Tabel kiri (mahasiswa) akan menjadi tabel master dan mencari nilai yang sama di tabel transaksi. Apabila ada mahasiswa yang tidak meminjam buku maka diberi nilai *NULL*.

C. *RIGHT JOIN*

Konsep *RIGHT JOIN* hampir sama seperti *LEFT JOIN* hanya yang menjadi master adalah tabel kanan (table 2)



Gambar 62. RIGHT JOIN

Jika table kiri tidak nilainya ada maka akan diisi nilai NULL

```
SELECT *
FROM table1
RIGHT JOIN table2
ON table1.field = table2.field;
```

Contoh, Mencari data dari table mahasiswa dan tranksaksi berdasarkan kolom NIM

```
SELECT *
FROM mahasiswa
RIGHT JOIN transaksi
ON mahasiswa.nim = transaksi.nim;
```

Hasil perintah tersebut seperti pada Tabel 37..

Tabel 37. pencarian data dengan LEFT JOIN

nim	nama	alamat	id_transaksi	nim	buku
21400200	faqih	bandung	1	21400200	Buku Informatika
21400202	anto	semarang	2	21400202	Buku Teknik Elektro

nim	nama	alamat	id_transaksi	nim	buku
21400203	dani	padang	3	21400203	Buku Matematika
21400206	senta	semarang	4	21400206	Buku Fisika
Null	Null	Null	5	21400207	Buku Bahasa Indonesia
Null	Null	Null	6	21400210	Buku Bahasa Daerah
Null	Null	Null	7	21400211	Buku Kimia

7 rows in set (0.00 sec)

Terdapat 7 transaksi peminjaman buku di perpustakaan. Bagi transaksi yang NIM mahasiswa tidak ada di table mahasiswa akan diberi nilai *NULL*

6.7. Cara Membuat *Stored Procedure*

Stored Procedure adalah sebuah fungsi berisi kode SQL yang dapat digunakan kembali. Dalam Stored Procedure juga dapat dimasukkan parameter sehingga fungsi dapat digunakan lebih dinamis berdasarkan parameter tersebut.

Cara penulisan *Stored Procedure*

```
DELIMITER $$
 CREATE PROCEDURE nama_procedure()
BEGIN
   kode sql
END$$
DELIMITER ;
```

Sedangkan untuk menjalankan *Stored procedure* adalah

```
CALL nama_procedure();
```

A. Hands ON

Sebelum mencoba menggunakan *Stored Procedure* kita harus mempunyai tabel yang siap digunakan.

Referensi contoh membuat table dan record dapat dibaca di materi DDL dan DML.

Buat tabel mahasiswa dan isi nilainya

```
CREATE TABLE mahasiswa
(
   nim INT(10),
   nama VARCHAR(100),
   alamat VARCHAR(100),
```

```
   PRIMARY KEY(nim)
);
```

```
INSERT INTO mahasiswa
VALUES
(21400200,"faqih","bandung"),
(21400201,"ina","jakarta"),
(21400202,"anto","semarang"),
(21400203,"dani","padang"),
(21400204,"jaka","bandung"),
(21400205,"nara","bandung"),
(21400206,"senta","semarang");
SELECT * FROM mahasiswa;
```

Hasil perintah tersebut seperti pada Tabel 38.

Tabel 38. Hasil membuat data tabel mahasiswa dan nilainya

Nim	Nama	Alamat
21400200	faqih	bandung
21400201	ina	jakarta
21400202	anto	semarang
21400203	dani	padang
21400204	jaka	bandung
21400205	nara	bandung
21400206	senta	semarang

7 rows in set (0.00 sec)

Membuat *Stored Procedure* **selectMahasiswa()** untuk m*end*apatkan seluruh NIM dan NAMA mahasiswa.

```
DELIMITER $$
CREATE PROCEDURE selectMahasiswa()
BEGIN
   SELECT nim, nama FROM mahasiswa;
END$$
DELIMITER;
```

Untuk memanggil *Stored Procedure* **selectMahasiswa()**

```
CALL selectMahasiswa()
Hasilnya adalah
CALL selectMahasiswa();
```

Hasil perintah tersebut adalah

Nim	Nama
21400200	faqih
21400201	ina
21400202	anto
21400203	dani
21400204	jaka
21400205	nara
21400206	senta

7 rows in set (0.00 sec)

Jadi, setiap ingin menampilkan NIM dan NAMA mahasiswa kita tidak perlu membuat kode SQL seperti biasanya. Cukup simpan kode SQL di *Stored Procedure* dan bisa kita panggil dan gunakan berulang-ulang *Stored Procedure* dengan Parameter. Kita juga memasukkan parameter di Stored Procedure agar menjadi lebih dinamis. Contoh, kita ingin membuat kode SQL untuk mencari data mahasiswa berdasarkan alamat.

```
DELIMITER $$
CREATE PROCEDURE alamatMahasiswa
(
  alamatMhs VARCHAR(100)
)
BEGIN
   SELECT *
   FROM mahasiswa
   WHERE alamat = alamatMhs;

END$$

DELIMITER ;
```

Di dalam nama *Stored Procedure* terdapat parameter **alamatMhs** dengan tipe data varchar. Saat masuk ke kode SQL yaitu mencari mahasiswa dengan alamat yang sama dengan parameter yang diinputkan.

Cara memanggilnya adalah

```
CALL alamatMahasiswa("bandung")
```

Hasilnya

```
CALL alamatMahasiswa("bandung");
```

Nim	Nama	Alamat
21400200	faqih	bandung
21400204	jaka	bandung
21400205	nara	bandung

3 rows in set (0.00 sec)

Dengan *Stored Procedure* alamatMahasiswa() kita mencari data mahasiswa yang berasal dari “bandung”.

B. DML dengan Stored Procedure

Stored Procedure tidak hanya bisa diterapkan di *Data Query Language* (DQL) tetapi juga *Data Manipulation Language* (DML)

Misal kita ingin memasukkan data mahasiswa dengan *Stored Procedure*.

```
DELIMITER $$
 CREATE PROCEDURE insertMahasiswa
(
  nimMhs INT(10),
  namaMhs VARCHAR(100),
  alamatMhs VARCHAR(100)
)
BEGIN
   INSERT INTO mahasiswa
   VALUES (nimMhs, namaMhs, alamatMhs);
END$$
```

```
DELIMITER;
```

Jika kita ingin memasukkan *record* baru di table mahasiswa maka akan ada 3 parameter saat memanggil *Stored Procedure* **insertMahasiswa()**.

```
CALL insertMahasiswa(21400207,"joni","jakarta");
Query OK, 1 row affected (0.05 sec)

SELECT * FROM mahasiswa;
```

Nim	Nama	Alamat
21400200	Faqih	Bandung
21400201	Ina	Jakarta
21400202	Anto	Semarang
21400203	Dani	Padang
21400204	Jaka	Bandung
21400205	Nara	Bandung
21400206	Senta	Semarang
21400207	Joni	Jakarta

8 rows in set (0.00 sec).

Selanjutnya jika ingin memasukkan data mahasiswa ke table mahasiswa tidak perlu membuat kode *INSERT* berkali-kali. Kita bisa gunakan *Stored Procedure* **insertMahasiswa()** untuk menggantikan proses *INSERT* yang biasanya kita gunakan. Jadi Stored Procedure sangat penting dan akan memudahkan dalam menggunakan kode yang ingin dieksekusi secara berulang – ulang.

6.8. Cara Membuat TRIGGER

TRIGGER adalah kumpulan kode SQL yang berjalan secara otomatis untuk mengeksekusi perintah *INSERT, UPDATE, DELETE*.

Biasanya *TRIGGER* akan dijalankan sebelum atau sesudah proses *INSERT, UPDATE, DELETE* (Perintah DML)

Cara penulisan *TRIGGER*

```
DELIMITER $$
CREATE TRIGGER nama_trigger
{BEFORE | AFTER} {INSERT | UPDATE| DELETE }
   ON nama_table
   FOR EACH ROW
BEGIN
   KODE SQL
END$$
DELIMITER ;
```

Untuk memulai menggunakan *TRIGGER* kita gunakan ***CREATE. TRIGGER*** dilanjutkan nama *TRIGGER* yang ingin dibuat.

{BEFORE | AFTER} adalah waktu *TRIGGER* akan dijalankan, apakah sebelum atau sesudah database dimodifikasi oleh perintah DML

{INSERT | UPDATE | DELETE} adalah perintah DML yang mengaktifkan *TRIGGER.*

Lebih detail waktu *TRIGGER* dijelaskan di Tabel 39.

Tabel 39. pencarian data dengan LEFT JOIN

Waktu *TRIGGER*	**Keterangan *TRIGGER***
BEFORE INSERT	*TRIGGER* dijalankan sebelum *record* dimasukkan ke database
AFTER INSERT	*TRIGGER* dijalankan sesudah *record* dimasukkan ke database
BEFORE UPDATE	*TRIGGER* dijalankan sebelum *record* dirubah di database
AFTER UPDATE	*TRIGGER* dijalankan sesudah *record* dirubah database
BEFORE DELETE	*TRIGGER* dijalankan sebelum *record* dihapus di database
AFTER DELETE	*TRIGGER* dijalankan sesudah *record* dihapus di database

ON m*end*efinisikan tabel yang mengaktifkan *TRIGGER.*

BEGIN END adalah pernyataan yang membungkus kode TRIGGER

Pastikan diawal gunakan **DELIMITER $$** dan diakhir dikembalikan ke ***DELIMITER ;*** seperti dalam membuat *Stored Procedure*.

Hands On

Pada *Hands-On TRIGGER* akan dibuat 2 tabel yaitu **table mahasiswa** dan **tabel log_mahasiswa.**

```
Table mahasiswa → menyimpan data mahasiswa
Table log_mahasiswa → menyimpan perubahan data mahasiswa
```

Jadi setiap ada perubahan data (*UPDATE*) alamat mahasiswa pada **tabel mahasiswa** maka akan disimpan di **table log_mahasiwa** tentang histori perubahan data alamat tersebut.

Dengan adanya *log* perubahan data mahasiswa maka akan memudahkan dalam melihat histori data mahasiswa yang pernah berubah dalam sistem.

Tabel mahasiswa

```
CREATE TABLE mahasiswa
(
   nim INT(10),
   nama VARCHAR(100),
   alamat VARCHAR(100),
   PRIMARY KEY(nim)
);
```

```
INSERT INTO mahasiswa
VALUES
(21400200,"faqih","bandung"),
(21400201,"ina","jakarta"),
(21400202,"anto","semarang"),
(21400203,"dani","padang"),
(21400204,"jaka","bandung"),
(21400205,"nara","bandung"),
(21400206,"senta","semarang");
```

Table log_mahasiswa

```
CREATE TABLE log_mahasiswa
(
   id_log INT(10) AUTO_INCREMENT,
   nim INT(10),
   alamat_lama VARCHAR(100),
   alamat_baru VARCHAR(100),
   waktu DATE,
   PRIMARY KEY(id_log)
);
```

Membuat *TRIGGER*

Kita akan menyimpan data perubahan alamat sebelum perintah *UPDATE* dijalankan.

```
DELIMITER $$
CREATE TRIGGER update_alamat_mahasiswa
   BEFORE UPDATE
   ON mahasiswa
   FOR EACH ROW
BEGIN
   INSERT INTO log_mahasiswa
   set nim = OLD.nim,
   alamat_lama=old.alamat,
   alamat_baru=new.alamat,
   waktu = NOW();
END$$
DELIMITER ;
```

Keyword OLD digunakan untuk mengambil data kolom di tabel yang lama sedangkan *keyword* **NEW** digunakan untuk mengambil data kolom di tabel yang baru.

Sekarang kita akan coba update alamat mahasiswa dengan NIM 21400200. Sebelum diupdate alamat mahasiswa dengan NIM 21400200 adalah “bandung”.

Kita ganti alamat “bandung” menjadi “surabaya”.

```
UPDATE mahasiswa
SET alamat = 'surabaya'
WHERE nim = 21400200;
```

Sekarang coba lakukan perintah SELECT untuk melihat **table log_mahasiswa**

```
SELECT * FROM log_mahasiswa;
```

id_log	nim	alamat_lama	alamat_baru	waktu
1	21400200	bandung	surabaya	2019-11-02

1 row in set (0.00 sec)

Oke, *record* baru secara otomatis telah ditambahkan ke **tabel.**

log_mahasiswa untuk mahasiswa dengan NIM 21400200 yang telah diubah alamat awal “bandung” menjadi “surabaya”

Sedangkan pada table mahasiswa alamat yang tercantum adalah alamat yang baru

```
SELECT * FROM mahasiswa WHERE nim=21400200;
```

Nim	Nama	Alamat
21400200	Faqih	Surabaya

DAFTAR PUSTAKA

Al-Masree, H. K. (2015). Extracting Entity Relationship Diagram (ERD) From Relational Database Schem. *International Journal of Database Theory and Application*, *8*(3), 15–26. https://doi.org/10.14257/ijdta.2015.8.3.02

Dewson, R. (2015). SQL Server Management Studio. *Beginning SQL Server for Developers*, 25–42. https://doi.org/10.1007/978-1-4842-0280-7_2

Kalaivani, S., & Shyamala, K. (2016). *A Novel Technique to Pre-Process Web Log Data Using SQL Server Management Studio*. *7*, 973–977.

Purwoko, H., & Sulaiman, H. (2021). *Penerapan Basis Data Relasional Pada*. 56–60.

Puspitasari, D., Rahmad, C., & Astiningrum, M. (2016). Normalisasi Tabel Pada Basisdata Relasional. *Jurnal Prosiding SENTIA | ISSN: 2085-2347*, *8*(1), 340–345.

Raharjo, S. (2012). *Constraint Basis Data Sebagai Fondasi yang Kuat Dalam Pengembangan Sistem Informasi*. *November*, 347–352.

Yuliansyah, H., Studi, P., Informatika, T., & Ahmad, U. (2014). Perancangan Replikasi Basis Data Mysql Dengan Mekanisme Pengamanan Menggunakan Ssl Encryption. *Jurnal Informatika Ahmad Dahlan*, *8*(1), 102982. https://doi.org/10.12928/jifo.v8i1.a2081

Perancangan Basis data (Database) adalah suatu proses untuk menentukan isi dan pengaturan data yang dibutuhkan untuk mendukung rancangan sistem. Tujuan Perancangan Database adalah untuk memenuhi informasi yang berisikan kebutuhan-kebutuhan user secara khusus dan aplikasi-aplikasinya.

Materi Buku Perancangan basis data ini meliputi Pengenalan database, bagaimana melakukan Normalisasi database dari NF, 1NF, 2NF, 3NF, Bagaimana membuat Relasi database, Membuat Entity Relationship Diagram (ERD).

Materi buku ini juga mencakup tentang penggunaan software Microsoft SQL Server Management Studio 2012. Penggunaan software ini untuk tingkat pemula, yang meliputi pengenalan software, cara melakukan instalasi software, membuat database dengan Graphical User Interface (GUI), penggunaan Query, bagaimana memahami Data Definition Language, Data Manipulation Language, Data Query Language, Data Control Language, Data Transaction Language, Penggunaan Join, membuat stored procedure, membuat Trigger

TENTANG PENULIS

Setiyowati, S.Kom., M.Kom

Lahir di Giriwoyo, Wonogiri, 25 Juni 1976, Dosen di STMIK Sinar Nusantara Surakarta. Lulus Program Sarjana Jurusan Teknik Informatika, Universitas Surakarta. Magister Komputer, Program Studi Teknik Informatika, Fakultas Teknik Industri, Universitas Islam Indonesia, Yogyakarta. Mengampu Mata Kuliah: Analisa & Perancangan Sistem Informasi, Sistem Basis Data, Manajemen Risiko. Email, setiyowati@sinus.ac.id

Sri Siswanti, S.Kom., M.Kom.

Lahir di Klaten, 15 Mei 1972. Salah satu dosen aktif di STMIK Sinar Nusantara Surakarta yang mengampu matakuliah Sistem Penunjang Keputusan, Multimedia Animasi, Manajemen proyek Sistem Informasi. Menyelesaikan jenjang S1 sistem Informasi dan S2 dari Universitas Dian Nusawantoro Semarang. Email: syswanty@sinus.ac.id